KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Technology

Pedestrian detection for real-life applications

Floris De Smedt

Supervisor:
Prof. dr. ir. Toon Goedemé
Prof. dr. ir. Tinne Tuytelaars, co-supervisor

Dissertation presented in partial
fulfilment of the requirements for the
degree of Doctor in Engineering Technology

November 2015

Pedestrian detection for real-life applications

Floris DE SMEDT

Examination committee:
Dr. Luc De Cooman, chair
Prof. dr. ir. Toon Goedemé, supervisor
Prof. dr. ir. Tinne Tuytelaars, co-supervisor
Prof. dr. Joost Vennekens
Prof. dr. ir. Luc Van Eyken
Dr. Stefan Schulte
Prof. dr. ir. Bart Vanrumste
Prof. dr. ir. Andrea Cavallaro
(Queen Mary University London, UK)

Dissertation presented in partial fulfilment of the requirements for the degree of Doctor in Engineering Technology

November 2015

Preface

If I have seen further, it is by standing on the shoulders of giants.

I. Newton

November 10th, at last the final presentation of my PhD research takes place. According to the information found on the Internet, obtaining the degree of "Doctor" requires having two main qualities, the capability to work independent, and to perform original scientific research. Although I hope having these qualities, and/or improved them over the last years, I have many people to thank who helped me reaching this point.

In the first place I want to thank my supervisor Toon Goedemé, who recruited me in the EAVISE research group to work on the SIVOL project, helped me finding an innovative research topic for my PhD and supported me on a daily basis working out my research ideas. He was a great help for improving the quality of my research by reviewing papers, participating in the research related discussions and creating a nice work environment, while always searching for funding to support, and extend, the EAVISE research group. Thanks to him I acquired a large experience between programming a convolution filter to segment weeds from soil, and developing an on-board multi-threaded pedestrian detection system on a UAV for following pedestrians. Evidently I also want to thank my co-supervisor Tinne Tuytelaars, who in the first years of my PhD performed the task of supervisor. The meetings we had where always an incentive to finishing the most recent ideas to demonstrate, and to be subjected to a critical opinion, which helped determining the next step(s) in my research trajectory.

I want to thank my examination committee for the valuable feedback. The notes and questions about my dissertation where of great help. I have tried to integrate them to my best ability, while finding a compromise with my own opinions.

Ook dank ik mijn collega's voor het dagdagelijkse plezier en de sfeer in het EAVISE team. De practical jokes, die al eens durfden te escaleerden tot bijvoorbeeld een stofzuiger op het dak, vormden een meerwaarde aan het werkleven die moeilijk in de industrie te vinden zou zijn. Samen op conferentie kunnen gaan was altijd leuker dan alleen, en ook de lokale pizza-industrie heeft kunnen welvaren bij de laat-avond sessies om de papers afgewerkt te krijgen.

Ik dank ook al de docenten die bereid zijn geweest hun kennis aan mij over te dragen. Het is door hun motivatie en tijd die ze spenderen aan het voorbereiden van lessen en labo's, dat ik met trots mag zeggen dat ik mijn master opleiding aan De Nayer heb behaald. De impact van een vak gedoceerd krijgen van iemand die door en door thuis is in het domein, en de motivatie heeft die kennis over te dragen, is niet te onderschatten. Het is omdat mijn laatste jaren aan De Nayer zo leerrijk waren, dat ik niet twijfelde om er nog wat jaren op te laten volgen.

Mijn dank gaat ook uit naar al de factoren die me hielpen mijn gedachten te verzetten van computers en onderzoek. Zo denk ik aan zaalvoetbalploeg Hak Strak, die me laat voelen dat ik geregeld aan sport moet doen, al eens voor de nodige discuties op het veld weet te zorgen, maar toch steeds als een vriendengroep weet af te sluiten. Er is uiteraard ook Scouts Lubbeek. Ondanks dat ik al enkele jaren de fakkel heb doorgegeven, is het altijd een genoegen verwelkomt te worden op de activiteiten. Maar ook de sporadische wandelingen langs de Vaart met het aangename gezelschap behoren tot deze categorie.

Ten slotte, maar daarbij minstens even belangrijk als al het voorgaande, wil ik ook mijn dankbaarheid laten uitgaan naar mijn ouders en broers. Papa en Mama, jullie hebben me altijd de kansen en de steun gegeven om mijn traject door mijn studies te vinden, al was dit niet altijd even eenvoudig. Sebastiaan, Valentijn en Tobias, onze interesses mogen dan wat verschillen, inzet, ambitie en passie voor wat we doen hebben we allemaal. Of het nu gaat over bosbeheer op een ecologische manier, de ontwikkeling van chips voor allerhande doeleinden, het leggen van groendaken of het bereiden van feestmaaltijden, het doet me allemaal breder kijken dan mijn eigen kennisdomein, en met de nodige bewondering. De steun (in al zijn vormen en maten), goede raad, of de gelegenheid "er gewoon eens van te kunnen klappen" die ik op zowel de leuke als de moeilijkere momenten van mijn familie heb gekregen, is een belangrijke reden dat ik de eindmeet van dit doctoraat heb gehaald.

Er zijn er uiteraard nog zoveel anderen die ik moet danken, maar ik denk dat al de lezers staan te trappelen om te weten wat ik in hemelsnaam allemaal heb gedaan de laatste jaren. Veel leesplezier.

Abstract

The presence of cameras in our surroundings grows every day, allowing object detection, and more specifically pedestrian detection, to be used in a broad amount of applications. Possible applications include surveillance applications, traffic safety, blurring pedestrians for privacy issues and human-robot interaction. In the last decade, pedestrian detection received a lot of attention as a research topic, which resulted in a broad amount of available techniques. In this dissertation, we focus on the applicability of these techniques, and propose a number of techniques to apply them in real-life applications.

Our main contributions are:

- We present a generally applicable technique to combine multiple pedestrian detectors, which we coined *The Combinator*, to considerably improve the detection accuracy, based on parameters describing each detector. We experimentally point out the accuracy benefit of this approach.
- We present a hybrid CPU/GPU implementation of the *Deformable Part Model* detector. This implementation exploits full parallelisation capabilities of both CPU and GPU. We combine this approach with a tracking-by-detection framework and a generalised ground plane estimation technique, to obtain a processing speed of 500 detections per second at high detection accuracy.
- We propose a technique to model a ground plane estimation as a first order linear function. We extend this technique such it becomes parameterizable by the inertial sensor values of a drone. This allows performing accurate real-time pedestrian detection from a flying UAV.

Beknopte samenvatting

Camera's vinden meer en meer hun aanwezigheid in ons dagelijks leven. Deze aanwezigheid zorgt dat object detectie, en in het bijzonder persoonsdetectie, gebruikt kan worden in een uiteenlopende groep van toepassingen. Mogelijke toepassingen zijn onderandere beveiligingsapplicaties, het onherkenbaar maken van personen voor privacy doeleinden en mens-robot interactie. In het laatste decennium heeft persoonsdetectie heel wat aandacht gekregen als onderzoeksdomein, waardoor er ondertussen heel wat technieken beschreven staan in de literatuur. In deze doctoraatsdissertatie leggen we de focus op de toepasbaarheid van deze technieken, en bespreken we verschillende technieken om de accuraatheid en snelheid van de bestaande technieken te verbeteren met het oog op toepasbaarheid.

De voornaamste bijdragen van deze doctoraatsdissertatie zijn:

- We stellen een techniek voor om meerdere persoonsdetectie algoritmen te combineren, *The Combinator*, om de detectie nauwkeurigheid aanzienlijk te verbeteren. De combinatie die we maken is gebaseerd op twee parameters die specifiek zijn voor elke detector. Experimenteel tonen we de nauwkeurigheidswinst van deze techniek aan.
- We ontwikkelden een hybride CPU/GPU implementatie van de *Deformable Part Model* detector. Hierbij wordt optimaal gebruik gemaakt van de parallellisatie mogelijkheden van zowel GPU als CPU. We combineren deze implementatie met een tracking-by-detection framework en een gegeneraliseerde benadering van het grondvlak om hoge accuraatheid aan 500 detecties per seconde te bekomen.
- We stellen een techniek voor om het grondvlak te modelleren als een eerstegraads lineaire functie. Deze techniek breiden we uit zodat deze parameteriseerbaar wordt aan de hand van de sensorwaarden van een UAV. Dit laat toe om real-time accurate persoonsdetectie uit te voeren van op een vliegende UAV.

Glossary

ACF	Aggregate Channel Features [28]
AdaBoost	Adaptive boosting, a machine learning technique
Caltech	California Institute of Technology, in this dissertation we refer to Caltech for a pedestrian detection dataset they made available [33]
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture, a language developed by Nvidia, which allows to use the GPU for data processing
CVPR	Computer Vision and Pattern Recognition, a yearly computer vision conference
DPM	Deformable Part Model [39]
ETH	Eidgenössische Technische Hochschule, in this dissertation we refer the ETH for the pedestrian datasets they made available [37]
EVW	Embedded Vision Workshop, a workshop focussing on embedded applications in computer vision, mostly held in conjunction with CVPR
FN	False negatives, missed detections, annotations that are not detected
FP	False Positives, detections that do not match an annotation
FPDW	Fastest Pedestrian Detector in the West [30]
FPPI	False Positives Per Image, the number of incorrect detections that on average is made per image while detecting. This is mostly used in combination with MR to indicate the accuracy of a detection algorithm.

FPPW	False Positives Per Window, the number of incorrectly classified window samples that on average is made. This is mostly used in combination with MR to indicate the accuracy of a detection algorithm. In [33] is pointed out however that this measure is flawed.
fps	Frames per second, the amount of images that can be evaluated per second by a certain algorithm
GPGPU	General-purpose computing on Graphics Processing Units
GPU	Graphical Processing Unit
HOG	Histograms of Oriented Gradients, in most cases we mean the HOG-SVM detector as proposed by [19]
ICF	Integral Channel Features [32]
IMU	Inertial Measurement Unit, the inertial sensors of a device used for, among other, stabilisation, determining the position, ...
LUT	LookUp Table
LUV	CIE 1976 (L*, u*, v*) colour space, a colour space adopted by the International Commission on Illumination (CIE) in 1976
MR	Miss rate, the amount of the annotations missed by the detector. It is used as a measure to describe how accurate a detector is, mostly in combination with FPPI (False Positives Per Image)
MT-DPM	Multi-Task Deformable Part Models, this detector, proposed in [94], improves the detection accuracy of the DPM detector by using a scale-independent feature space
NMS	Non-Maximum-Suppression
PCA	Principal Component Analysis
PDF	Probability Density Function
SIFT	Scale-invariant Feature Transform, a key-point description technique [63]
SVM	Support Vector Machines, a machine learning technique
TP	True Positives, detections that match an annotation
UAV	Unmanned Aerial Vehicle, in this dissertation we use this term when we refer to a quadrocopter or a hexacopter

VGA	Video Graphics Array, an image resolution of 640x480
VJ	Viola and Jones, we refer to VJ for the face detection technique [89]
WSPD	Warp-Speed Pedestrian Detector, the hybrid CPU/GPU implementation of the *Deformable Part Model* detector [25]

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pedestrian detection has endless possibilities in many application areas. An evident application domain is surveillance. The number of cameras in public places such as train stations, shopping malls and streets grows each year, with security in mind. According to [13], there are currently 287,236 surveillance cameras active in Belgium, with an increase of almost 10% in the last year. Note that these numbers do not include the cameras used in cars, mobile devices, ... A direct response in the cause of an incident, however, requires manual observation of the video stream, which is in most cases economically infeasible. This means that these cameras are mostly used to capture evidence material after the incident. We believe however that an improvement and more efficient use of object detection, and more specifically pedestrian detection, can be of help as a pre-processing step for continuous analysis of these video streams.

When we look into the domain of traffic safety, we see that more and more modern cars are provided with a system to detect possible collision with pedestrian, cyclists and other cars [18]. But larger vehicles, such as trucks, still cause to many casualties which could be avoided with the proper technology [86, 87]. Similar systems could be applied for improving safety around agricultural vehicles when working off-road.

Also in e-health applications could pedestrian detection play an important role. Just one example is the detection of falling elderly people to automatically trigger an alarm. This kind of supporting technology allows the elderly people to live longer in their familiar environment [91].

This broad applicability of human detection, and object detection in general, makes this topic receive a lot of interest over the past decade. A number of

promising algorithms have been developed over time, and improvements on both accuracy and speed are published each year. Most of these algorithms are primarily tested in lab surroundings though, and/or are only available in Matlab, which makes them difficult to apply in real-life applications.

The two main properties that determine the usefulness of detection algorithms are the *evaluation speed*, and the *detection accuracy*. Evidently does the evaluation speed depend on the computation platform, e.g. modern computers with state-of-the-art processors allow to perform a multi-threaded evaluation at high processing speed, but such systems are not practically feasible when pedestrian detection has to be performed on-board from a flying drone (a case study we will discuss in detail in chapter 7). The detection speed is also dependent on the application. In general the assumption can be made that a throughput (the number of frames processed per time unit) should be at least equal to the frame rate of the camera (the number of frames captured by the camera per time unit). Note however that the task of pedestrian detection may only be a part of a larger system. E.g. when pedestrian detection is used in cars to avoid collision, also the time required to generate an alarm and triggering the breaking system has to be taken into account. This traffic safety application shows that we have to make a distinction between a high throughput and a small latency, in which the latter measures the time between capturing the frame and the moment the detection results are available to be used by a subsequent step. In chapter 4, where we discuss the use of parallelisation to improve detection speed, we discuss the difference between these two in more detail.

The *detection accuracy* describes how well a certain (pedestrian) detection method performs. The accuracy is always measured on a specific dataset containing mutual appearances of pedestrians, where each of these pedestrians is marked as ground truth. Based on this ground truth, the accuracy can be measured as the comparison between the number of these marked pedestrians that are found, and how many incorrect detections are returned by the detection method. Note that this accuracy measure is strongly dependent on the dataset. E.g. it is far easier to detect a pedestrian standing before a high contrasting wall, than to detect a soldier using camouflage clothing in a forest.

Not only does the kind of clothing plays an important role, but also the camera resolution, the amount of motion blur (e.g. when captured from a driving vehicle) and the size the pedestrian has to be detected on. In chapter 2 we discuss the accuracy measurement in more detail, and which techniques are used to compare the accuracy between different detection methods. Note that the detection accuracy is related with the detection speed, since the use of a complex model (such as used by the *Deformable Part Models* detector [39]) or the use of features that require more computational power (such as using techniques as *Local Decorrelation Channel Features* (LDCF) [67]) will increase

the computational cost of the pedestrian detection method, and as a consequence decrease the detection speed.

The research question we try to answer with this dissertation is "How can we apply current state-of-the-art pedestrian detection techniques in real-life applications?". To answer this, we focus both on techniques to improve the detection accuracy, by combining multiple detection approaches, and the detection speed without accuracy loss. Evidently is applying the proposed techniques to improve the detection speed of detector methods individually, also beneficial when they are used as a combination to improve accuracy.

In section 1.1 we give our main contributions of this dissertation. In section 1.2 we describe a small selection of example applications for which we retrieved a request from industrial partners, for which pedestrian detection plays a crucial role. To conclude our introduction, we give an outline of this dissertation in section 1.3.

1.1 Contributions

In the context of this dissertation we created a framework that includes multiple pedestrian detection algorithms, that allows to apply the proposed techniques independent of the detector to use, to allow a fair comparison of the detectors. The main contributions we propose in this dissertation are:

- We present a generally applicable technique to combine multiple pedestrian detectors, which we coined *The Combinator*, to considerably improve the detection accuracy, based on parameters describing each detector. We experimentally point out the accuracy benefit of this approach.
- We propose a general applicable technique to integrate scene knowledge in the pedestrian detection algorithm to improve detection speed. The techniques we describe are based on temporal information, in the form of a tracking-by-detection framework, and a ground plane estimation function as a first order linear function.
- We describe multiple methods to exploit multi-core architecture, including multi-core CPUs and GPUs, to improve the detection speed without constraining the applicability of the algorithm or the detection accuracy.
- As a first case study, we combine the use of GPU and multi-core CPU hardware with heavy search space reduction to obtain a speed of 500 detections per second in the application of the detection of vulnerable road users in the blind spot area of trucks.

- As a second case study we combine pedestrian detection with pedestrian tracking to follow pedestrians from a flying UAV, using an on-board implementation. For this we propose an extension of the ground plane constraint, to make it parameterizable with the data of the inertial sensors of the UAV.

1.2 Example applications

Since this is a PhD in the faculty of Engineering Technology, it is important the work is applicable for industrial related cases. Below we give some cases based on requests the EAVISE research group received from industrial partners related to pedestrian detection. For each of these cases we investigate the requirements.

1.2.1 Blurring of pedestrian's faces in mobile mapping images

Due to a public tender, many land surveying companies have contracts for photographing the entire public space of Flanders. This creates huge amounts of data that contains privacy-sensitive material. Up to now, the annotation of the people's faces to be blurred is done manually, boiling down to a big cost. This makes it an ideal application for an automatic person detection algorithm. Figure 1.1 shows an example image for this application. The resolution of these images is very large, in this case 8000x4000 pixels.

Since the processing can be performed offline, this application has no strict real-time constraint, although speed improvements are always beneficial. Based on figure 1.1, we can indicate a large opportunity to reduce the searching space for this application, since half of the image area is consumed by air, which will not contain pedestrians.

1.2.2 Automatic capturing of web lectures and presentations

During the capturing process of web lectures and presentations, a pan-tilt-zoom camera will be instructed to automatically follow the lecturer, such that no camera man is necessary. This application introduces a mild real-time constraint, but missing the person for a few frames does not induce a problem. The challenge in this application is the presence of occlusion, which requires the use of multiple models (face, full body, upper body, ...) to be combined.

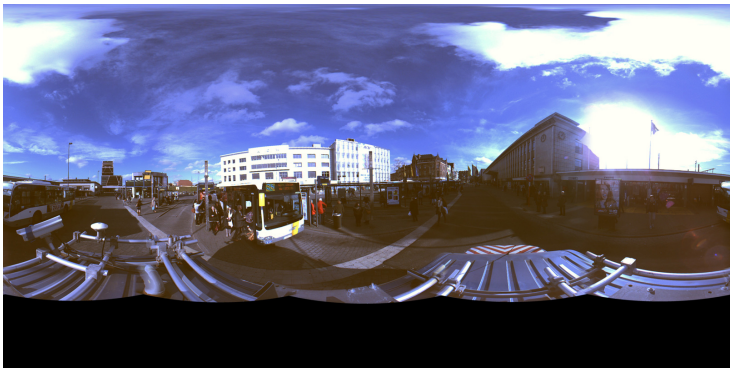


Figure 1.1: An example image of mobile mapping which requires the faces of pedestrians to be blurred.

Figure 1.2: Automatic following of the lecturer during web lectures. Taken from [52].

1.2.3 Detection of pedestrians around AGVs and in the blind spot area of trucks

A more challenging application is the detection of pedestrians around AGVs and in the blind spot area of trucks. Accuracy is of high importance to avoid false detections while detecting all vulnerable road users. Note however that the cost of missing detections, which could lead to deadly accidents, is higher than that of generating false alarms, although the latter makes the system less pleasant to use. At least equally important as the accuracy is the real-time constraint (10-15fps). The goal of this application is to improve the safety of pedestrians around the vehicles, by notifying the driver of their presence, for example by using a small screen which indicates where the pedestrian is located. A more challenging goal can be to even brake the vehicle automatically when detecting a pedestrian. On top of both a high reliability and real-time processing, the camera view can be challenging. Due to the use of wide-angle lenses, the camera view may be distorted, as can be seen in figure 1.3.

Figure 1.3: Example image from a blind spot camera on a truck [87].

1.3 Thesis outline

In **chapter 2**, we give an introduction on how the task of pedestrian detection is commonly approached. We give an extensive overview of the evolution in the state-of-the-art on pedestrian detection in the last 15 years, both concerning speed and accuracy. We give a detailed overview on how pedestrian detection algorithms are compared using generally accepted techniques, and some conditions we make to compare algorithms fairly. We finalise this chapter with an overview of the detectors we use throughout this dissertation for our experiments, and compare them according to the evaluation methodology we describe before.

In **chapter 3** we present a technique (*The Combinator*) we worked out describing how to optimally combine multiple pedestrian detection algorithms to improve detection accuracy [26], which we apply on our implementations using the C++ framework we developed in [21]. We give a detailed overview on how we cluster detections of different detectors, such that the detections of each cluster can be combined into a single score, using a weighted sum, for which we use a confidence value, indicating how trustworthy a detector is, and a complementarity value which indicates what the additional value of a detector is as part of the combination.

In **chapter 4** we exploit the capability of parallelisation on modern hardware to improve the detection speed. In contrast to the speed improvements we will discuss in chapter 5, these speed improvements are independent of the application. We discuss how evaluating the layers of the feature in parallel can benefit the processing speed, a technique we also applied in [21] to improve the detection speed of our combination approach of chapter 3. Since a naive implementation of this parallelisation approach does not scale well when increasing the number of threads used, we propose an improvement technique for a better balance in workload over the threads, which allows a more efficient use of the available hardware. We compare the efficiency of these implementations. Both of the these parallelisation techniques focus on decreasing the latency of the detection process, such that a faster response on the detection result is possible. We compare the parallelisation of feature layers, with evaluating multiple images in parallel, which increases the throughput but does not decrease the latency. Such an approach can be used when off-line processing is allowed, such as in the mobile mapping application we discussed in section 1.2.1, where the task is to blur the faces of pedestrians in a very large amount of images. Next to multi-core CPU based parallelisation, we describe how the feature pyramid of the *Deformable Part Model* detector can be evaluated on GPU (Graphical Processing Unit) using the OpenCL language [23].

In **chapter 5** we integrate scene knowledge to reduce the search space of detectors, and thereby improve the detection speed. In contrast to the speed improvements obtained in chapter 4, the techniques we discuss here use information from the scene. As a first approach, we demonstrate the influence of reducing the amount of layers in the feature pyramid, which we can do if we know beforehand at which scales pedestrians can occur. Next we describe how temporal information can be integrated as a tracking-by-detection framework, such that for most locations in the image, we will only search for pedestrians when one of the active tracks predicts the presence of a pedestrian. To initialise the tracks, we use *Initialisation Regions* at locations pedestrians could appear in the scene, in which we perform the detection algorithm at multiple scales. We used a very similar approach in [86, 25]. As a final technique to integrate scene knowledge, we make use of a ground constraint, by modelling the relation between a y-position in the image and the height in pixels of the pedestrian that could appear there, as a first order linear function. To compensate for the variation of the height between pedestrians and possible movement of the camera, we use two different offsets. This technique allows to drastically decrease the search space of the detection process [22].

We combine multiple of the described techniques as part of two case studies we worked out. The first case study is described in **chapter 6**, and focuses on the detection of pedestrians at a very high speed in the blind spot of trucks. We

use two speed improvements to obtain this high speed. We propose a hybrid CPU-GPU implementation where we combine our GPU implementation of the feature pyramid of DPM [23] with a CPU implementation of the model evaluation, which on its own has a speed-up of 12.7 times over the original Matlab implementation of this algorithm. On top of that we use a strong reduction of the search space by applying a tracking-by-detection framework with the Warping Window approach [87], a generalisation of the ground plane constraint which allows to cope with the heavy distorted frames of the blind spot camera (for more details, see section 5.6).

As a second case study, which we discuss in **chapter 7**, we develop an on-board system to follow pedestrians using a UAV. This chapter is based on a publication [22], which was co-authored by D. Hulens. The embedded system we use is selected according to his previous work [53] based on its computational power and weight criteria. To improve the detection speed, we propose a technique that makes the first order linear model of the ground constraint parameterizable by the inertial sensor data of the UAV. The location of the pedestrian in the image is determined by a combination of a colour based particle filter, which allows to overcome frames without detections, and a pedestrian detection algorithm which is used to keep the particle filter focussed on the pedestrian. The difference between the centre of the image, which is the place with the smallest chance of losing the pedestrian, and the location determined by the particle tracker, is used to steer the UAV with the intervention of a software based PID control loop.

We conclude this dissertation in **chapter 8** with an overview of the discussed techniques. We also give an extensive overview of possible future work on our research.

Chapter 2

Pedestrian detection

2.1 Introduction

Object detection is a sub-domain of computer vision with the goal of finding a class of objects in an image. Pedestrian detection can be performed by using the same techniques, although the used object-model is trained on pedestrians. An important distinction exists between **object detection**, meaning to detect a class of objects, e.g. *all pedestrians in the image*, and **object recognition**, where the object is identified, e.g. *that particular person*. Object detection can be the first step for object recognition tasks. Accurate and fast object detection is necessary in a broad range of applications such as automotive safety, surveillance and industrial automation. In this PhD study, we mainly focus on pedestrian detection, although the techniques used are also applicable for object detection in general. Due to the wide applicability of pedestrian detection, it has been a subject of great interest in literature for many years.

In this chapter we give an overview of different kinds of pedestrian detection techniques, of which a selection where applied during this PhD study. Although these algorithms have a similar design to distinguish the presence of pedestrians from the background, they all have unique properties which make them feasible in different situations. All the detectors we use are brought together into a single framework. This allows to apply the optimisations we described in papers for a specific detector, and which will be described in the following chapters, to all detectors.

In section 2.2 we will give an overview of the commonly used technique to approach pedestrian detection. In section 2.3 we will give an overview

of literature about pedestrian detection. The literature describes multiple optimisations towards both speed and accuracy of these algorithms, which we will utilize in later chapters. Section 2.4 explains the evaluation methodologies we apply in this dissertation.

In section 2.5.1 to 2.5.4 we will give a more detailed description of the *Histogram of Oriented Gradients* (HOG), *Deformable Part Models* (DPM), *Integral Channel Features* (ICF), *Aggregate Channel Features* (ACF) and *Squared Integral Channel features* (SqrtICF) detectors respectively. These are the algorithms we worked with during this PhD. In section 2.6 we will compare these algorithms both for speed and accuracy.

2.2 Pedestrian detection overview

Distinguishing pedestrians, and other objects, from the background is a challenging task, mostly solved in the same sequence of steps. In this section we give an overview of these returning steps, which are visualised in figure 2.3.

The first challenge is to find image features with high distinctiveness between properties of the object and the remainder of the image. The stronger this distinctiveness, the easier it will become to correctly find all the occurrences of the object in the image. An important property of this feature is the invariance to a changing environment, such as for example illumination changes. A widely used feature for this specific example is the use of gradients between neighbouring pixels.

The next step is to describe what the object we are searching for looks like in the used feature representation. In the remainder of this book, we will use the term *object model* for this object representation. This model is commonly obtained by using a large set of samples (both of the object, which are the positives, and from background, which form the negatives), each converted into the used feature format. A machine-learning algorithm such as AdaBoost, Support Vector Machines, Neural Networks, ... can then be used to learn an object model that forms the best balance between containing sufficient detail to distinguish the object from the background, and generic enough to cover inner class variation.

The model is compared with each location in the image. The similarity between the pre-trained object model and the image features indicates the probability of the object being present at that location. The process of looping over all the locations in the image (mostly performed in 4px or 8px spacing intervals) is commonly known as the *Sliding Window* approach. A threshold is used to define

(a) Scale-space pyramid (b) Feature pyramid

Figure 2.1: A visualisation of a scale-space pyramid and the corresponding feature pyramid.

the boundary between detections and background. This threshold determines the required similarity between the image features and the model, to be treated as a detection.

The next challenge is to cope with the multiple sizes at which the object can be present in the image. As we will see later, the object size in the image is dependent on the real-world size of the object, and the distance from the camera. To cope with this, a scale-space pyramid is created, where each layer is formed by rescaling the image with different scale ratios. The scale ratio that is used between two layers is commonly expressed as the number of scales per octave, so the number of layers before the resolution is divided by two. The sliding window approach will be performed on the features calculated on each layer of the scale-space pyramid. Figure 2.1 visualises a scale-space pyramid and the corresponding feature pyramid.

The last step is to filter out the detections related to the same object appearance. The small steps taken in the sliding window process has as a consequence that each presence of an object will be found multiple times in the same area. Therefore we perform *Non-Maximum-Suppression* (NMS), which, as the name implies, will prune all detections which are not a local maximum. A good NMS-algorithm will prune all superfluous detections, but will ensure that each object is still covered by one bounding box. This is a challenging task since multiple appearances of the object can be close together (e.g. pedestrians in a crowd). Traditionally, NMS is based on the *Intersection over Union* (IoU) criterion, shown in equation 2.1, where BB_1 and BB_2 represent the bounding boxes for which IoU is calculated.

$$IoU = \frac{area(BB_1 \cap BB_2)}{area(BB_1 \cup BB_2)} \tag{2.1}$$

For most evaluation benchmarks [33, 34, 36], the IoU between the detection and the ground truth annotation has to be at least 0.5 to be treated as a correct detection. In the process of NMS, when detections have a IoU value higher as a certain threshold, only the one with the highest score will be kept. This process is implemented by sorting the list of detections based on score, such that performing NMS can be performed by evaluating the list from begin to end. There can be made a distinction between *exhaustive* and *greedy* methods, where for greedy methods a detection that has been pruned will no longer be able to prune other detections, which makes the process less computationally intensive, but less pruning is performed. In the addendum [31] to [32], the authors reviewed these NMS choices to optimise the detection accuracy of the *Integral Channel Feature* detector. Figure 2.2 visualises the effect of these different approaches. Here they also propose a novel measurement technique (equation 2.2), by using the minimum area of both detections instead of the union (marked by a * in figure 2.2). As we can observe the different approaches require different overlap threshold values to obtain an optimal accuracy value.

$$Overlap = \frac{area(BB_1 \cap BB_2)}{min(area(BB_1), area(BB_2))} \tag{2.2}$$

2.3 Related Work

Due to the wide applicability of pedestrian detection in a variety of applications, including traffic, surveillance, robotics and safety, there has been a lot of research on this topic.

In 1996, Ojala et al. [69] proposed the use of LBP features as a texture measure. In a local neighbourhood of pixels, for example 3x3, the value of each pixel is compared with the value of the center pixel. When its value is higher or equal, it is marked by a 1, when it is smaller it is marked by a 0. The feature value for a particular pixel can be described as a sequence of eight bits this way. An example is shown in figure 2.4. To improve the robustness of this technique, a histogram of the pattern occurrences (possible sequences of bits) of the LBP features over a small area can be used. The simplicity and speed at which this feature can be calculated, next to its distinctiveness, makes it a valuable feature to work with. In [1] for example, Ahonen et al. proposed to use this feature for face recognition, where a sample is classified based on a nearest neighbour classifier using Chi square as a distance measure. They show the superiority of the use of LBP features over commonly used techniques, such as PCA, for this purpose.

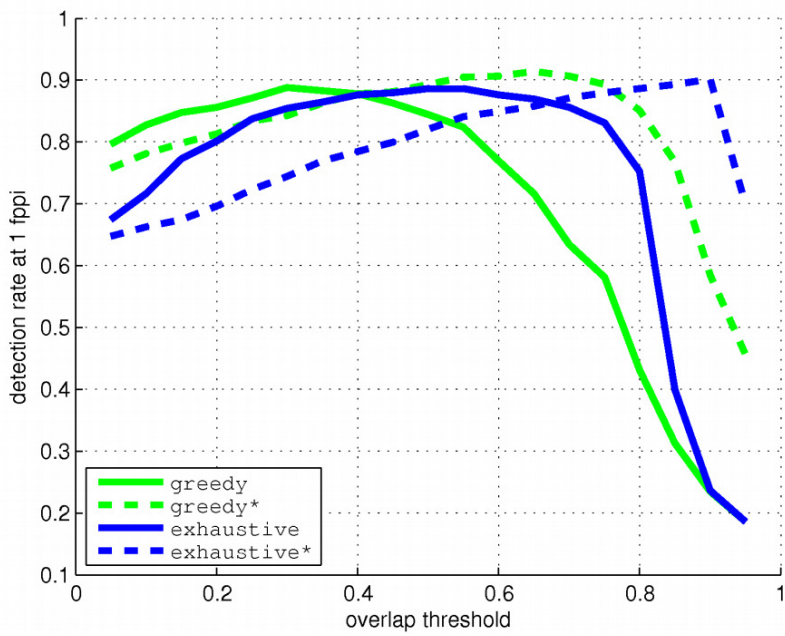
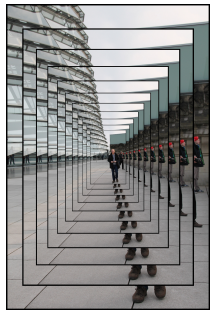


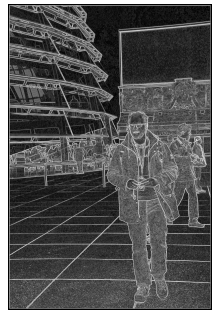
Figure 2.2: Comparison of the False Positives Per Image (explained in section 2.4) in function of the overlap threshold for different NMS-approaches. The techniques using the alternative measure are marked with *. Taken from [31].

In 2001 Viola and Jones [89] proposed the use of *Haar-like features* to calculate features for real-time face detection. Using these *Haar-like features*, of which a few examples are shown in figure 2.5, feature values are calculated by combining the sums of the luminance in rectangles, e.g. the bright region(s) minus the dark region(s). By using an integral image, the sum of any rectangular area can be efficiently determined independent of its size, which allows considerable speed. To obtain a strong classifier to distinguish faces from the background, a model is trained using AdaBoost as a linear combination of these features. Each feature of the model is a combination of a specific *Haar-like features* at a specific location on the face, and an associated threshold that indicates the boundary between background and a face. To improve detection speed, the features of the model are grouped into stages, where each successive stage is trained only on the training samples that are classified as a face by the previous stages. During evaluation, after each stage a certain threshold has to be reached to continue the evaluation, which allows early rejection of the sub-windows that look nothing like a face.

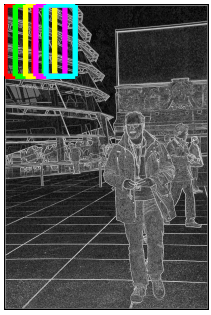
(a) Input image.



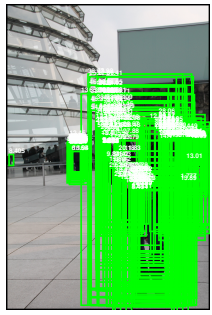
(b) Creation of the scale-space pyramid.



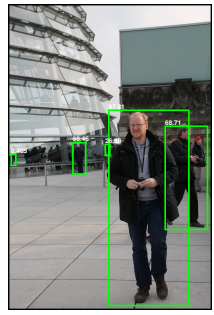
(c) Feature calculation for each layer of the feature pyramid



(d) Sliding-window evaluation on each layer of feature pyramid



(e) Detections before Non-Maximum-Suppression.



(f) Final detection results on the frame.

Figure 2.3: An overview of the common object detection approach.

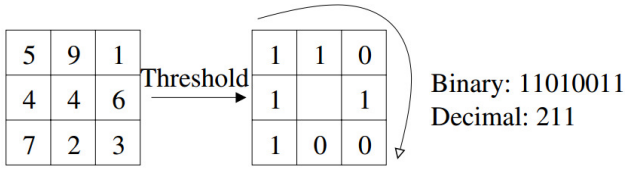


Figure 2.4: An example of how to calculate LBP features. The value of the eight surrounding pixels is compared to the value of the centre pixel. Taken from [1].

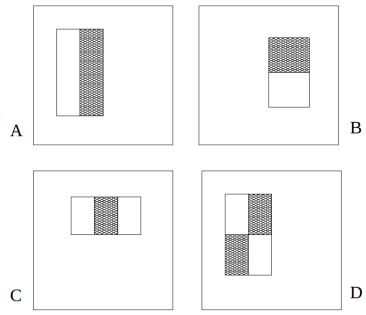


Figure 2.5: Examples of Haar-like features, the feature values are calculated as the difference between the sum of the pixels covered by the bright and dark region(s). Taken from [89].

The *HOG-SVM* detector, was proposed by Dalal and Triggs in 2005 [19]. As an image feature, it uses *Histogram of Oriented Gradients* (HOG) of the pixels in the image, such that each bin of a histogram represents a specific orientation of the gradient. The influence of a specific pixel gradient in a bin is proportional to the magnitude of the gradient vector. To improve accuracy, additional normalisation steps are performed, e.g. to avoid that one orientation will dominate the histogram. The resulting features are fed into an SVM classifier. The model used is 64x128 pixels, and includes a 16px padding around the pedestrians annotations at each side to take also some background into account. This has a clear beneficial impact on accuracy, as is shown in the original publication [19]. Datasets from more realistic conditions showed room for improvement on both accuracy and speed of this detector. However, even today their work is used as a reference in new benchmarks.

2.3.1 Part based pedestrian detection

To increase the accuracy, essentially two different approaches were proposed. A first approach is to make the model more complex, of which the work of Felzenszwalb et al. [42, 39] with their *Deformable Part Models* (DPM) detector is only one example. The DPM detector is based on the HOG-features as used in the HOG-SVM detector we described before [19]. The rigid HOG model is extended with part models, which embed details of certain parts e.g. the limbs of a person, into the model. The detection score at each location is the combined score of the rigid root model, which models the object as a whole, and the part models. The position of each part is allowed to deviate relative to the position of the root model, although this comes with a part-dependent

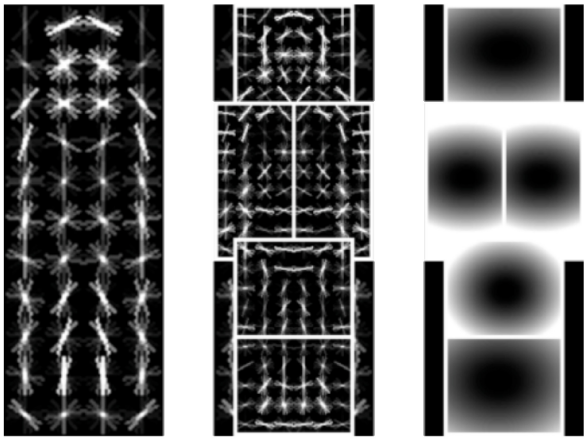


Figure 2.6: Model as used by the *Deformable Part Model*-detector for the detection of pedestrians. Left: root model, middle: part models, right: allowed position deviation relative to the root model.

deformation cost. This allows a pose variation of the object to detect. Figure 2.6 visualises the model used for the detection of pedestrians which is trained on the INRIA dataset [19].

The model is trained by using a Latent Support Vector Machine, which allows the training process to determine the most optimal position of the parts relative to the position of the root model. This avoids that the parts require to be annotated separately. The result is a set of convolution filters, each responsible for detecting one of the parts. The resulting detection score is the sum of the convolution of each model-part with the image features.

Using the parts, the DPM detector takes a lot of information of the pedestrian (or other object) into account. At low resolutions however, this detailed information is not always sufficiently available. To address this problem Park et al. proposed the MultiresC detector [73] in which a low resolution rigid template, similar to the HOG-SVM detector we described earlier, is used for low resolution detections, while the DPM detector is used for high resolution detections. This approach drastically improved the detection accuracy. In 2013 Yan et al. [94] tackled the low performance of the DPM detector on low resolution differently. They map the features from different resolutions to a common subspace with a resolution aware transformation. The DPM-model is then trained in this common subspace, such that is will equally work well on high and low resolutions. They coined this detector MT-DPM (MultiTask-DPM).

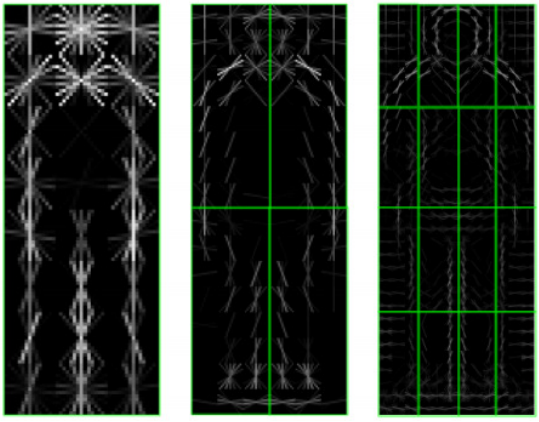


Figure 2.7: Model as used by the *Coarse-to-fine* approach for the detection of pedestrians. The model is evaluated from left to right only on the highest scoring position of the previous (coarser) filter. Taken from [75].

The use of multiple part filters comes with a heavy computational cost for evaluating this type of model. In literature multiple improvements have been proposed to improve this detection time. Felzenszwalb et al. [41] proposed the use of a cascaded approach similar to [12]. After the evaluation of each filter (both root and part), the combined score until then is compared to a pre-determined threshold. If this threshold is not reached, the location in the image is no longer considered as possibly containing the object, and therefore the model evaluation at that location is stopped. Viola and Jones used a similar approach in their work on face detection [89], although here the pruning thresholds are determined after the model is trained instead of being part of the training procedure. The use of this cascaded approach obtains a speed-up of over an order of magnitude for pedestrian detection.

According to Pedersoli et al. [75], the cost of the detection is not dominated by finding the correct placement of the parts relative to the root-model, but by the number of part-to-image comparisons. Therefore they propose the use of a coarse-to-fine approach, where the model is hierarchically structured in multiple levels, such that each level contains more detail (higher resolution) information as the previous. By evaluating this model from the lowest resolution model (coarse) to the more detailed (fine), only on the most promising placement of the filters, the search process has a speed-up of an order of magnitude, and when combined with the cascade approach even two orders of magnitude. Figure 2.7 visualises a pedestrian model using this approach.

In 2012, Dubout and Fleuret proposed the use of the Fourier transform of the HOG-features, which allows to calculate the evaluation of the model as a dot product instead of a convolution. This technique, in combination with evaluating the layers of the feature pyramid in parallel, allowed a speed-up of a factor 7 over the original DPM detector [39] for pedestrian detection. Since applying the Fourier transform of the calculated HOG-features takes a lot of the computation time, this approach becomes more effective when multiple models are evaluated on the same image. Although this approach is less effective for speed-up improvements as compared to the cascade approach on average, the evaluation time becomes deterministic and so independent of the image content. On top of that, by avoiding incorrect early pruning which can happen in a cascaded approach, a slight accuracy gain can be obtained, although the cascaded approach is effective enough to make this difference negligible.

Sadeghi and Forsyth [80] proposed the use of vector quantization for the HOG features. Using a training set, 256 clusters are determined using k-means clustering for the HOG-features. The dot-product between each cell of the model and each of the clusters is pre-calculated and stored in a look-up table. When the HOG features of an image are calculated, the nearest cluster is determined for each cell, which forms an approximation of the real feature values. This allows to encode each cell as a cluster number, for which the dot-product with model cells can be obtained from a LUT. This reduces the computational complexity from 32 multiplications and 32 additions, to 1 lookup per cell. The accuracy of this approach is dependent on the amount of clusters used, since the distance between the real features and the centres of the clusters defines the error that is made. Using 256 clusters has a negligible accuracy loss, while a speed-up of two orders of magnitudes could be obtained. In a subsequent publication [81] finding the cluster for each cell was optimised by performing it in two steps, where in each step only the best out of 16 clusters has to be found.

In 2014, the same authors of [94] proposed a drastic speed improvement of their MT-DPM detector [93]. They proposed three techniques: (1) by limiting the rank of the root filter, they ensured it can efficiently be evaluated as 1D evaluations (2) based on the crosstalk technique [29], they proposed *neighbourhood aware DPM*, such that many hypotheses in this cascade can be aggressively pruned, according to the first order approximation of the stage scores by their neighbourhoods, instead of explicitly computed and (3) instead of calculating the gradient magnitude and orientation, it is replaced by indexing operations into look-up tables (LUT) since there are only a finite number of possibilities of gradients and orientations. Using these optimisations, a speed-up of a factor 4 over the previously discussed techniques, so four orders of magnitudes over the original detector [39], is realised.

2.3.2 Channel based pedestrian detection

A second technique to improve detection accuracy over the HOG-SVM detector [19], is to enrich the features used for pedestrian detection, by using for example colour information next to gradient information. This is done by Dollár et al. [32] for their *Integral Channel feature* (ICF) detector. The combination of six gradient orientations and three colour channels (LUV) turns out to reach the best performance. Figure 2.8 visualises the calculated channels on an example image.

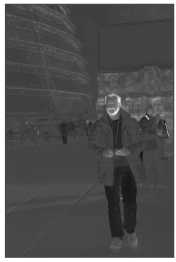
To reduce the computational load of the detector, the type of wavelets used are rectangles (first order features) instead of a combination of rectangles (higher order features) as was used by Viola and Jones. The features that are used in the final object model are selected using AdaBoost, which selects them from a random generated pool of 30 000 possible rectangles inside the model window. In the original publication, a model was trained using 2000 stages, where each stage represent a 2-level decision-tree of the possible feature values (each associated with a certain rectangle on a specific location). Also here does the calculation of the feature values benefit from the use of an integral image. As an addendum to their original paper [31], they proposed an improvement by rescaling the channels before calculating the integral image, which benefits evaluation speed, while having a negligible impact on detection accuracy.

In 2013, Benenson et al. [7], an evaluation of every step of the training process of the ICF detector was performed, resulting in the *Roerei* detector, which is to the best of our knowledge the most accurate detector evaluated on the Caltech pedestrian dataset [33], while trained on the INRIA dataset [19]. One of the optimisations they proposed was the use of all possible rectangles inside the model window, as opposed to the selection of 30 000. This huge amount of possible feature values makes the training process too computation intensive to be feasible however. As a compromise, a detector was proposed where all possible squares inside the model could be used to obtain feature values instead, coined the *SqrtChnFtrs* detector. The number of possible squares inside the model window is evidently a lot less than all possible rectangles, which makes the computational cost of the training process feasible, while still allowing to cover the complete area of the person.

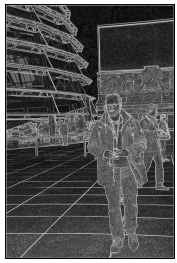
In 2013, Park et al. [74] proposed the use of motion information as an additional feature. The motion information is obtained by using a coarse-scale optical flow, based on the technique of Lucas-Kanade [64], to align the images of a sequence. This new information source was both used on top of the HOG-LUV channels and trained with a boosting approach such as the ICF detector [32], and on top of a HOG-SVM part based detector as described in [39]. For both

(a) L channel of LUV.

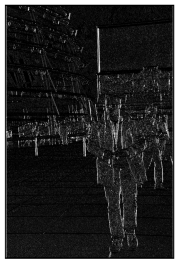
(b) U channel of LUV.



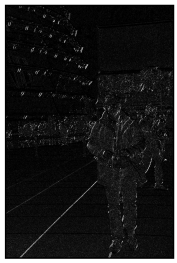
(c) V channel of LUV.



(d) Gradient magnitude



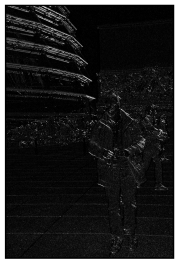
(e) 0° gradient channel.



(f) 30° gradient channel.

(g) 60° gradient channel.

(h) 90° gradient channel.



(i) 120° gradient channel.

(j) 150° gradient channel.

Figure 2.8: The channels used as feature by the object detectors from the ChnFtrs family: 3 LUV colour channels, the gradient magnitude and 6 gradient orientations.

of these detector types, the use of motion information formed a reliable feature that improved the accuracy drastically. Inspired by the work of Bharath et al. [50], Woonhyun et al. [67] solved a shortcoming of the decision trees used in the commonly used boosting approach for object detection. For efficiency reasons, the decision trees with orthogonal (single feature) splits are used, which in contrast to oblique (multi feature) decision splits, do not achieve top performance on correlated data. By proposing a transform that removes local correlation of the data, a high accuracy can be achieved while still using the fast orthogonal decision trees as weak classifiers. In practise the use of the LDCF (Local Decorrelated Channel Features) are implemented as four linear filters applied on the ten HOG-LUV channels, resulting in 40 channels to select features from. Benenson et al. [8] performed an overview of the techniques applied on pedestrian detection to improve detection accuracy over the last decade. By combining a SqrtChnFtrs detector [7] with the motion information of [74] and the local data decorrelation of [67], they propose the *Katamari* detector, which achieves very high accuracy on the Caltech testset [33]. Recently, Zhang et al. [98] created an additional accuracy improvement on this combination of techniques by using higher order features (combination of rectangles), as were used in the Viola and Jones face detector [89].

Although the use these rigid detectors have a speed advantage over the more complex part-based detectors on their own, multiple speed improvements have been described in literature. In 2010 Dollár et al. [30] proposed the use of feature approximation. Instead of calculating the features at all layer of the feature pyramid, only a limited amount of features is completely calculated from image data, as was done for all layers before, while the others are approximated. This approach leads to a speed-up of an order of magnitude while only having a small drop in detection accuracy. In [28] the effect of feature scaling on object detection is analysed in more detail, and evaluated for different detector implementations. In this work, the authors present a new detector, coined *Aggregate Channel Features*, which uses only squares sampled on a rigid grid to obtain feature values. When the size of these squares equals the width of the pixel block used to calculate the HOG-features on, the cost of obtaining feature values is reduced to a pixel-lookup, avoiding the requirement of using an integral image. By using a strongly optimised implementation to calculate the HOG-LUV features, the resulting detector is one of the fastest available today, even on single core CPU hardware. They use the work of [3] to allow training a new model in only a few minutes, as opposed to hours or even weeks.

In 2013, Benenson et al. [6] used an approach where an object model is obtained for each scale of the feature pyramid as a training step. While detecting, this only requires to calculate the features for one scale, instead of a the whole pyramid, on which all these models can be evaluated. Since training all these

models from scratch would require a lot of computational power, only a reduced number of models is trained this way, while the others are approximated based on the technique we described previously [30]. This particular optimisation allowed to detect at twice the speed as the baseline detector. Combined with a GPU implementation and a ground constraint, which we describe in section 4.3 and 5.2 respectively, they reached a speed of 100 frames per second.

In [29] another kind of cascade-approach is proposed as *Crosstalk cascade*, based on the idea that detection responses at nearby locations are correlated. By exploiting this correlation, large areas of the image can be fastly rejected such that their approach reaches speed-ups of factors between 4 and 30 over the original algorithm. Note that the techniques we described earlier focus on improving the speed of a single window (one specific location), while here the relation between locations is exploited.

2.3.3 Deep learning

A recent trend in computer vision is the use of convolutional neural networks (convnets or CNNs). In deep learning, of which convnets is only one technique, the learning is performed in multiple cascaded layers, so each layer works further on the previous. The larger the amount of layers, the "deeper" the network. A key feature of deep learning is the capability of handling very large amounts of data and the end-to-end learning. In contrast to previous mentioned techniques, the features are not hand designed, but determined as part of the whole image-to-detection task, and so work on the raw image data. Although the success stories of this technique [60, 79], it is still not thorough studied in the task of pedestrian detection.

In [71], the authors used CNNs to tackle the feature extraction, deformation, occlusion and classification as one problem with success. CNNs are also applied to learn context around pedestrians [96]. Note that all mentioned approaches do not use convnets for the complete detection pipeline, but use a "traditional" pedestrian detection technique to prune the amount of detection windows to evaluate. Recently, Hosang et al. [51] applied CNNs to pedestrian detection without explicitly modelling the problem into the network, using *out-of-the-box* network architectures (the "small" CifarNet and the large AlexNet). Hereby they obtained results that are very competitive with the current state-of-the-art of traditional pedestrian detection approaches. But also here a traditional pedestrian detection approach, (SqrtChnftrs) [7], is used to reduce the number of candidates to 100 per image to be classified by the CNN. They claim a processing time of 3ms per window.

Although further research on this topic could lead to better accuracy results than traditional pedestrian detectors, CNNs have a high computational requirement which makes them infeasible to be used in a sliding window manner. Therefore they depend still on "traditional" pedestrian detection approaches to obtain reasonable processing times. This makes the window proposal technique the most computational expensive task.

The dependence on "traditional" pedestrian detection makes the remainder of this work, which focuses on improving such approaches, still very beneficial, since these can be used as a window proposal technique. Also techniques to speed up traditional pedestrian detection techniques, such as search space reduction, may be transferable to CNNs based detectors.

2.4 Evaluation methodology

In this section we will explain how pedestrian detection algorithms, and object detection algorithms in general, can be fairly evaluated and compared.

There are two properties that are important for a detector, which is the evaluation speed and the detection accuracy. The evaluation of the detection speed seems rather straightforward. The algorithm has to be evaluated on each frame of the dataset, and the detection speed, which is mostly expressed in *frames per second* (fps), can be found by measuring the total required time to evaluate the dataset and dividing this time by the amount of frames processed. It is more complex though.

As we noted earlier, one of the steps of the detection process is evaluating a pedestrian model at each location and each scale in an image. This makes the evaluation time dependent on the size of the model, since using a larger model implies less locations to be evaluated per layer. On the other hand, the size of the model, in combination with the image resolution, also defines the minimum object size that can be found. For example, if a model with a height of 100px is used, an up-scale to twice the resolution is required to find objects with a height of 50px. This up-scale implies also the requirement of calculating the feature values over a larger area. To compare detectors fairly, they should be evaluated with the same requirements, and thus the same range of detection heights. In chapter 5, in the explanation of using a ground plane constraint, the comparison is made between two *Aggregate Channel Feature* (ACF) models. The first is trained on a model height of 50px, while the other has a height of 100px. Since the requirement of that comparison is detecting pedestrians with a minimum height of 50px, using the latter model comes with the up-scale requirement. Although the smaller model is more complex (more stages, and

each stage is a decision tree with more levels), this smaller model has the speed advantage. Therefore, in this dissertation we will evaluate the detectors on multiple image resolutions.

Next to the detection speed, the detection accuracy of a detector is of great importance. The detection accuracy describes how *good* the detector performs in the task it was trained for: detecting people (or other objects) in images. To measure this, a dataset is used, where each occurrence of the object to detect is annotated (marked with a rectangle) as ground truth. Figure 2.9 visualises an example frame from the *Caltech* dataset [33] with the annotated pedestrians marked by a rectangle.

The accuracy of a detector is commonly evaluated by comparing the accuracy at different thresholds, which allows to find out a good balance between the number of correct detections that are found, and the amount of mistakes that are made. In the work of Dalal and Triggs [19] about their HOG-SVM detector, the accuracy is approached as a classifier, since in the end it is an SVM classifier that for each window in the *Sliding Window* approach decides if it is a pedestrian or not. The set of samples to classify is constructed from windows centred on the annotations (ground truth) of pedestrians, which should be classified as pedestrian by the detector, and samples from images without pedestrians, which should be classified as non-pedestrians or background. The *Miss Rate*, which is the share of ground truth annotations that are not correctly classified as pedestrians, is plotted versus the *False Positives Per Window* (FPPW), which indicates the share of the window samples that on average are incorrectly classified as pedestrians. An example of such a curve is given in figure 2.10, which shows the performance on the INRIA dataset to compare the effect of different window sizes using the HOG-SVM detector. The optimal point on this curve is at the bottom left (coordinate (0,0)) where everything is classified correctly.

In the work of Dollár et al. [33] is pointed out that the use of the FPPW accuracy measure is flawed, since this *per window* measure can not accurately predict the *per image* performance of a detector, which is how a detector is used in practice. In this dissertation we only use *per image* measures, which we explain below.

To measure the accuracy of the detector, it is evaluated on each frame of the dataset. And on each frame the found detections are compared with the annotations (ground truth) on that frame. According to [38, 34, 33] an IoU (Intersection over Union) overlap of at least 50% between a detection and an annotation is sufficient to be counted as a correct detection, and thus a *True Positive* (TP). Note however, that for each annotation there can only be one detection match. All detections that do not match an annotation are False

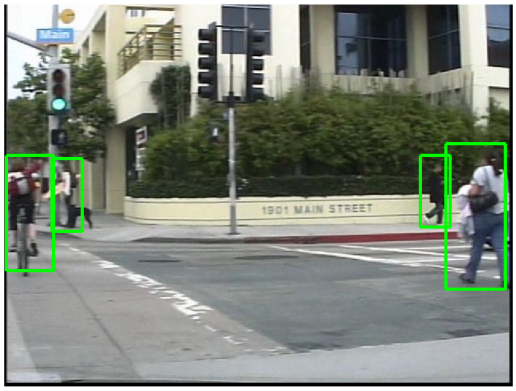


Figure 2.9: Example image from the Caltech dataset [33] with ground truth annotations shown.

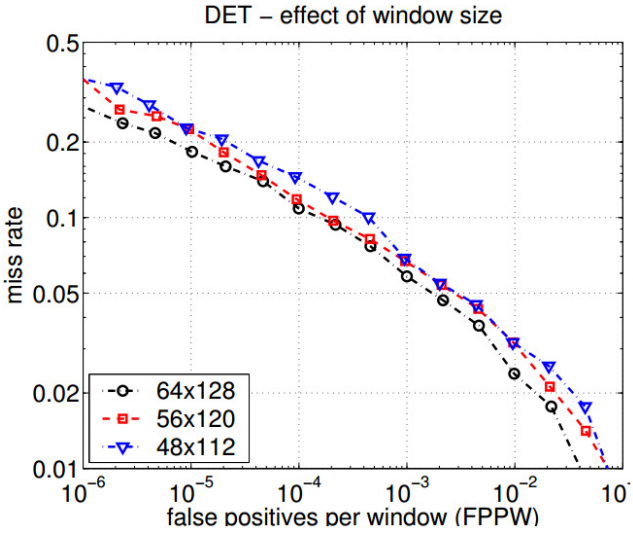


Figure 2.10: Example FPPW curve to evaluate the resulting accuracy when using different window sizes. Taken from [19].

Positives (FP). The annotations that are *not* covered by a detection are the *False Negatives* (FN). Evidently, the goal is to have the TP as high as possible, and both the FN and FP as low as possible.

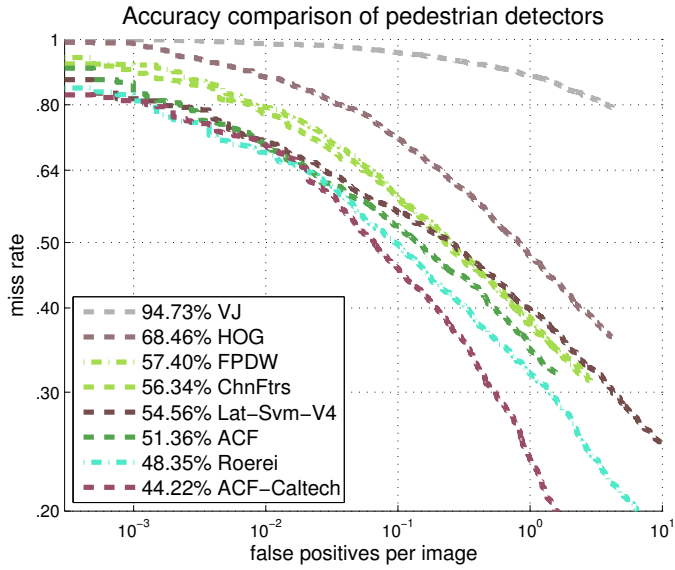
The amount of detections on a frame is dependent on the threshold that is used while performing the detection process. The higher the threshold is set, the less detections will be returned, but when the detector is correctly trained, most of the found detections will be TP. The accuracy of an algorithm can be visualised by evaluating the algorithm over a range of thresholds. The two types of plots we use in this dissertation are the *FPPI vs. Miss Rate* curve and the *Precision vs. Recall* curve, which are the established methodologies to evaluate pedestrian detection algorithms. *FPPI*, which stands for False Positives Per Image, indicates the average number of incorrect detections (FP) made per evaluated frame. The *Miss Rate*, calculated by equation 2.3, indicates the share of annotations that is not detected. Both of these measures need to be as small as possible. The second type of curve we use in this dissertation is the *Precision vs. Recall* curve, where the *Precision*, which is calculated using equation 2.4 and shows the share of detections which is found that is correct (TP), is plotted versus the *Recall*, which is calculated using equation 2.5 and shows the share of annotations that is found.

$$Missrate = \frac{FN}{TP + FN} \tag{2.3}$$

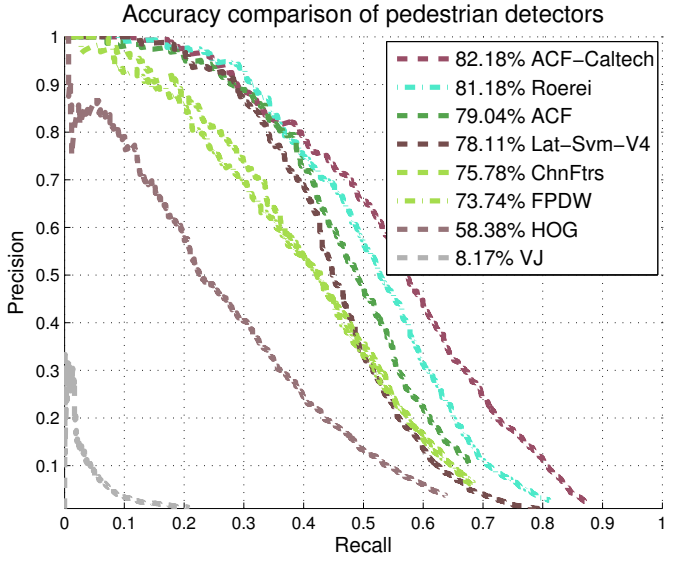
$$Precision = \frac{TP}{TP + FP} \tag{2.4}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.5}$$

Dollár et al. [34, 33] proposed a framework to compare a lot of pedestrian detection algorithms based on different datasets. Both of the curves we discussed above are possible outputs of the evaluation framework, which they made publicly available. This framework has support for different types of *experiments*, of which the complexity is dependent on the range of pedestrian sizes that has to be detected, the (maximum) amount of occlusion that may be present and the amount of IoU overlap required to be considered a correct detection. For our experiments we always use the *Reasonable* setting, which means a height of 50px and up, minimum 65% of the pedestrian must be visible (not occluded) and at least 50% IoU is required. Figure 2.11 show the *Miss Rate vs. FPPI* and *Precision vs. Recall* curve for some of the algorithms of this evaluation framework.



(a) Example *MR vs. FPPI* curve.



(b) Example *Precision vs. Recall* curve.

Figure 2.11: Examples of the two types of accuracy curves we use in this dissertation.

Each point on these graphs represents running the algorithm using a specific threshold. When this threshold is low, most of the pedestrians in the dataset will be detected, which gives a low miss rate and a high recall, but also more non-pedestrians (false detections) will be detected, which means a higher amount of false positives per image (FPPI) and a lower precision. This results in a point at the right of both curves, while at the other end a high threshold will leave out most false detections (a low FPPI value and a high precision), but could also lead to missing part of the annotated pedestrians, and thus a higher miss rate and a lower recall. The point to use on these accuracy curves, which determines the threshold to use, is dependent on the requirements of the application. The optimal point on a *Miss rate vs. FPPI* is at the lower left corner (coordinate (0,0)), and for a *Precision vs. Recall* curve at the upper right corner (coordinate (1,1)). Keep in mind that most detectors use a cascaded detection approach to allow early pruning while detecting. When a lower detection threshold is used, the thresholds in the levels of the cascade need to be altered accordingly to impose less pruning. This will also influence the detection speed of the detector.

2.5 Detector implementations

In this section we give an overview of the detectors we have implemented and used for the experiments in the scope of this PhD.

2.5.1 Histograms of Oriented Gradients

As we discussed in the related work section, the use of HOG-features forms a steady foundation for pedestrian detection in general. The HOG-SVM detector [19] is still used as a comparison in benchmarks. For our work, we made use of the available implementation in the OpenCV computer vision library [14], although we replaced the NMS algorithm with the one used by the ACF detector [28] (*greedy**). In our implementation, we shrink the bounding boxes to compensate for 16px padding used when training the HOG model. In the remainder of this dissertation, we will refer to HOG-SVM detector as the HOG detector for short.

2.5.2 Deformable Part Models

An implementation of the (cascaded) *Deformable Part Model* detector [41, 42] is made publicly available for Matlab [40]. Parts of the code are implemented using mex (an API for using c-code) for performance reasons. We ported this

algorithm to a full C++ version. This avoids the requirement of installing Matlab on each device we want to run the algorithm on, and comes with an improved detection speed, as we show in section 2.6. We used variants of this code in several publications [23, 25, 21]. In section 4.7 we describe how we ported the calculation of the feature pyramid to OpenCL to be evaluated on GPU hardware.

2.5.3 Integrated Channel Features

The *Integral Channel Feature* detector forms a cornerstone for a lot of research over the years [6, 7, 30, 28, 98], which makes it indispensable for this PhD study.

The implementation(s) we made of these algorithms are based on the channel feature code released as part of the Piotr Dollár Matlab toolbox [27], and are part of the *Aggregate Channel Features* Matlab implementation [28]. The C++ implementation we created for this detector was in the scope of our publication "Open framework for combined pedestrian detection" [21] and is based on the description of the original paper.

2.5.4 Aggregate Channel Features

This recently published detector forms a combination of accuracy with high processing speed, which makes it an optimal detector to be used for applications where high processing speed is necessary. Therefore we ported this algorithm from a Matlab implementation, which already contains large portions of C-code for performance reasons, to a full C++ implementation, which we used as part of our work for autonomous drone steering to follow pedestrians in "On-board real-time tracking of pedestrians on a UAV" [22]. Details of this work will be given in chapter 7.

2.5.5 Squared Channel Features

Based on the SqrtChnFtrs-detector proposed by Benenson et al. [7], where all possible squares inside the model window are used to obtain feature values, we extended our ACF implementation, which we describe in section 2.5.4. In our implementation, we do not use all possible squares though, but only those with dimensions that are a multiple of the applied shrinking factor. In our implementation we also use the approximation of channel features as described in section 2.5.4 to still obtain high processing speed.

2.6 Comparison

As described in the previous section, we have created a variety of detector implementations: HOG, ICF, DPM, SqrtICF and ACF. Here we will evaluate them based on accuracy and speed, by using publicly available datasets. For the task of accuracy evaluation, we make use of the evaluation framework proposed by Dollár et al. [33], which was designed with comparing detection accuracy of pedestrian detectors in mind. Since the publication of this framework, over 40 detectors are evaluated on the datasets for which the results are made available, which allows a broad comparison. For the evaluation on the *Town Centre* dataset, we use our own Matlab based evaluation framework.

In the course of this dissertation, we use *greedy** with an overlap criterion of 0.65 to perform NMS on the detections of all detectors. Note that, although this is the default NMS approach for *Aggregate Channel Feature* detector [28], the *Deformable Part Model* detector [40] uses *exhaustive* with an overlap criterion of 0.4 as default. Changing the *NMS* approach for *DPM* has no notable effect on detection accuracy though.

The datasets we will use for our comparison are:

- **Caltech-USA** This dataset [33] is captured using a forward looking camera in a driving car. All the pedestrians in the resulting video stream are annotated with additional occlusion (the share of the pedestrians that is not visible) information. The images are divided into 11 sets, and each set is divided into multiple video-sequences. The first 6 sets are assigned for training, while the last 5 are used for evaluation. Due to the large amount of images (121,465 in the evaluation sets alone), evaluation is classically performed on only 1 in 30 frames, reducing the amount of images to 4024. We will use the naming convention *Caltech-30* for this manner for evaluation.

- **ETH** This dataset [37] is divided into three sequences, resulting in 1804 images in total, taken with a stereo setup mounted on a stroller. For evaluation only the left frames are used.

- **Town Centre** As the name implies is this dataset [10] recorded in a busy town centre street. It has a resolution of 1920×1080 and is recorded at 25 fps. It has the ground truth of 71500 hand labelled head locations (from which the pedestrian's bounding box can be estimated), with an average of 16 people visible at any time. This makes it a very crowded dataset compared with the others. In this dissertation, we mainly use this dataset to demonstrate our tracking-by-detection framework in chapter 5.

2.6.1 Accuracy

We evaluated our algorithms on the aforementioned datasets, using the image and annotation data available in the evaluation toolbox. The comparison with the original algorithms is performed using Caltech-30, since the complete evaluation data (Caltech-1) is not available for all detectors. This comparison is shown in figure 2.12. The percentages in the legend are calculated as an average miss rate based on a number of sample points. From this we can see that our own implementations are close approximations of the original implementations these are based on. Sometimes we even obtain a slight accuracy improvement, although this is rather coincidental and due to small differences in implementation (image rescaling, rounding, floating point precision,...). To obtain smoother curves and a more complete evaluation, we evaluated our own implementations also on Caltech-1, which is shown in figure 2.13. The comparison of our algorithms on the ETH and Town Centre datasets is shown in figure 2.14 and 2.15 respectively.

Based on these graphs, we can make some remarks. The first is that although DPM is not a top-performing algorithm on Caltech, it obtains a large accuracy gain over the others on the ETH and Town Centre dataset. We assume this accuracy gain is due to the pose-variation that *DPM* allows. A second observation we can make is that although our implementation of SqrtICF is trained with a larger feature-pool compared to ACF, this does not guarantee an improved performance.

2.6.2 Speed

Next to the accuracy of an algorithm, the processing speed may be of great importance when selecting an algorithm for an application. To evaluate the algorithms fairly, our evaluation takes two remarks into account:

- The model size determines the smallest detectable pedestrian in the image. For example, to obtain detections using the DPM detector, which uses a 120px high pedestrian model, from 50px upwards, requires up-scaling the image with a factor of 2.4. Evidently this imposes longer processing times. Therefore we will give the processing time for multiple image resolutions.
- Most algorithm use a cascade approach which allows early pruning during model evaluation. Evidently does a more greedy setting (higher thresholds) lead to faster processing, but also leads to a higher miss rate (point more to the left on the accuracy graphs). For a fair speed comparison, we will evaluate all algorithms to detect at 0.1 FPPI on Caltech-30.

The evaluation we perform is on all the images of Caltech-30, using only single threaded CPU implementations. In later chapters we will go deeper into improving detection speed using dedicated hardware such as GPUs, parallelisation on multi-core CPUs and by reducing the search space. Since the hardware system to evaluate on plays an important role for evaluation of processing time, we evaluate all algorithms on the same system, which may lead to different results as reported in our papers (and other literature). The system we use in this PhD dissertation has two octacore E5-2687W processors running at 3.1GHz, 64GB 1600MHz DDR3 memory and an Nvidia quadro K5000 graphical card with 4GB of on-board memory. We will explicitly note when the results we show are obtained on another hardware configuration.

In table 2.1 we give an overview of the calculation time of the pedestrian detection implementations. Next to the calculation time we also give the model size used for evaluation, since the smallest possible detection depends on this. From this table we can observe a large speed improvement of the more recent detector architectures, based on the work of Dollár et al. [28], over the others.

Technique	Model size	640x480	1280x960	1536x1152
Ours-HOG	96px	2.35 fps	0.58 fps	0.40 fps
Ours-ICF	100px	3.48 fps	0.68 fps	0.46 fps
Ours-DPM	120px	1.72 fps	0.46 fps	0.32 fps
Ours-ACF	100px	31.94 fps	6.9 fps	4.75 fps
Ours-SqrtICF	100px	26.65 fps	6.78 fps	4.57 fps
Matlab-DPM	120px	1.5 fps	0.39 fps	0.30 fps
Matlab-ACF	100px	25.88 fps	5.65 fps	4.21 fps
FFLD	120px	1.42 fps	0.39 fps	0.26 fps

Table 2.1: Speed comparison of the pedestrian detection algorithms, all running on a single core.

As we could observe in table 2.1 is our cascade implementation faster as FFLD [35], which performs the convolutions with the model in the Fourier domain as a dot product, although the difference is not that big. As we compare the peak memory-use when processing VGA-resolution images of both detectors however, as we did in [21], we observe that *Ours-DPM* uses "only" 110MB of memory, while the FFLD implementation uses 805MB. Note that here we evaluate VGA resolution images. When an evaluation of smaller pedestrian sizes is required, up-scaling the image is necessary, and by consequence more memory will be used. On a desktop system, the peak memory-use may seem unimportant, but for embedded devices, or very high image resolutions, this forms a problem.

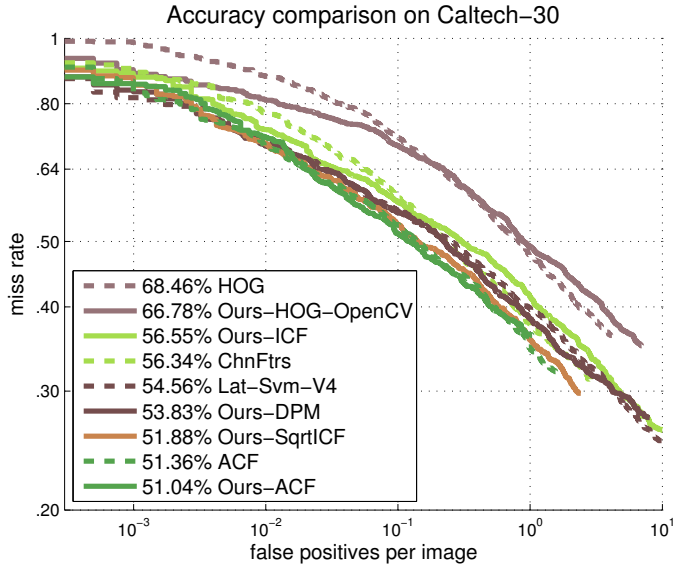


Figure 2.12: Evaluation on Caltech-30.

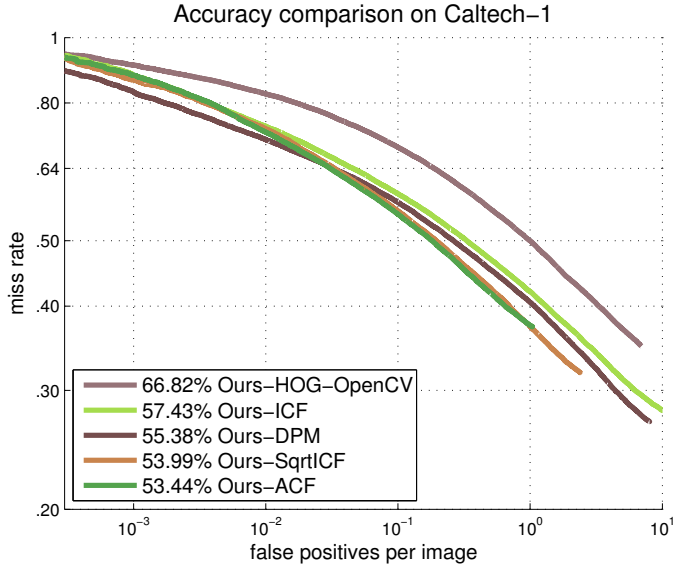


Figure 2.13: Evaluation on Caltech-1.

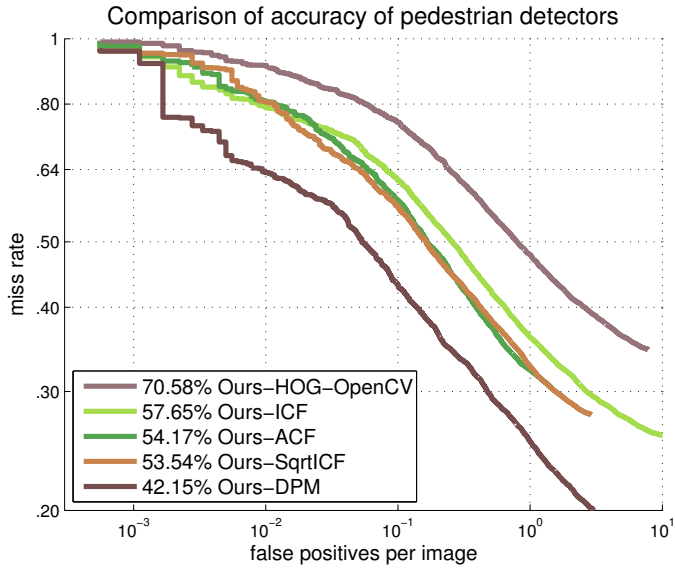


Figure 2.14: Evaluation on ETH.

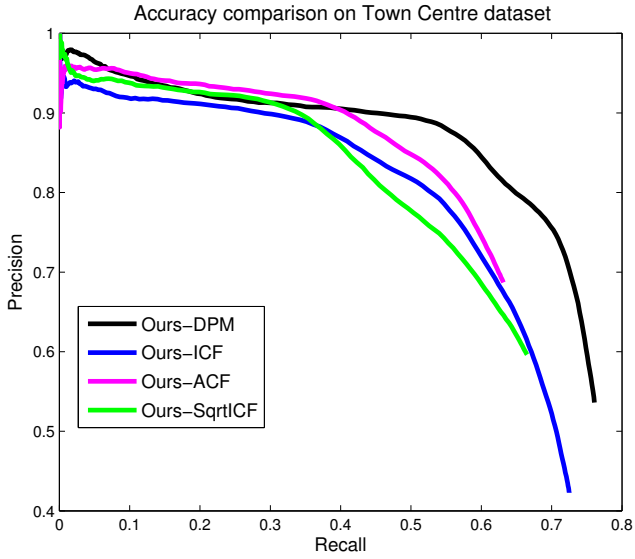


Figure 2.15: Evaluation on Town Centre.

2.7 Conclusion

In this chapter we gave a thorough overview of pedestrian detection. After giving an overview of how object detection (including pedestrian detection) is classically performed, we gave a more detailed overview of our implementations of existing pedestrian detection algorithms we use through the remainder of this dissertation. For a fair comparison in processing speed, we ran all algorithms on the same hardware system, and for multiple image resolutions. Here we saw that our fast C++ implementations reach the same accuracy as the original publications they are based on. Finally we compared the accuracy of our implementations using multiple public available datasets. Here we saw that both the absolute and the relative performance between detectors differ based on the kind of image data we are processing. So being the best on one datasets does not guarantee being the best on other datasets. In the next chapters we focus on improving both the accuracy and detection speed of these detectors.

Chapter 3

Combined pedestrian detection

In this chapter we focus on the improvement of detection accuracy by combining the results of multiple detectors. To combine them, we worked out a weighted sum method that takes both the *confidence* (how good a detector is, independently of the others) and the *complementarity* in a combination (what the additional value is of detectors over each other) into account.

The content of this chapter is a combination of multiple publications. In 2014, we published "The Combinator: optimal combination of multiple pedestrian detectors" (ICPR 2014) [26]. This was a joint publication with K. Van Beeck. Using the evaluation framework of [33, 34], we created a Matlab implementation to optimally combine the detection results of the Matlab implementation of *Deformable Part Models* [41], *Integral Channel Features* [32] and *Histogram of Oriented Gradients* [19]. Note that this implementation did not contain the detection algorithms themselves, but only used the detection results. The focus of this work was the improvement of accuracy, without considering speed for a complete implementation.

In "Open framework for combined pedestrian detection" (VISAPP 2015) [21] this combination approach was applied on our own pedestrian detection implementations of these three pedestrian detection algorithms, which enabled the use of a combination as an independent detector. By porting our Matlab implementation of [26] to a complete C++ implementation, we solved an issue with the clustering of detections (described in section 3.3.1), and solve it as a graph problem, to guarantee that the accuracy of the resulting

combination is independent of the order in which the algorithms are executed. Therefore, the results we will present in this chapter will be obtained by the implementation of the latter publication, and extend the experiments using all of our implementations we presented in section 2.5.

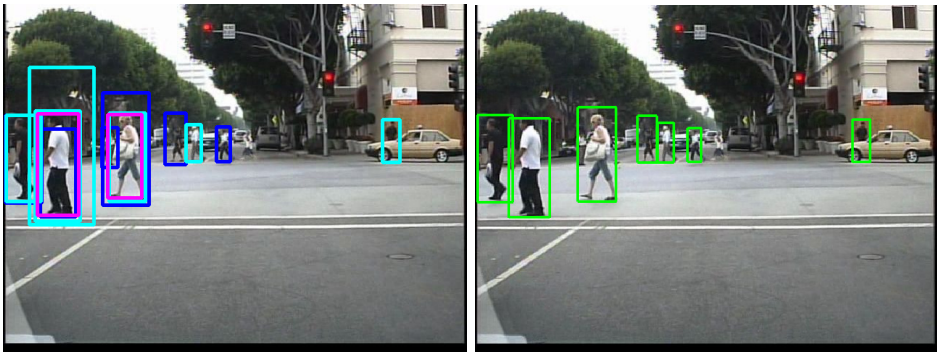
The concept of combining detectors based on a *Confidence* and a *Complementarity* value, and possible techniques how to measure them, was contributed by Floris De Smedt. Working out the details and implementing this idea was done in co-operation with K. Van Beeck, which is published as "The Combinator: optimal combination of multiple pedestrian detectors" (ICPR 2014) [26]. The work for the sequel publication "Open framework for combined pedestrian detection" (VISAPP 2015) [21], discussing practical use of a combination of detectors, was fully contributed by Floris De Smedt.

3.1 Introduction

As we described in chapter 2, pedestrian detection is an active research topic in the last decade. Indeed, state-of-the-art algorithms achieve excellent accuracy results on challenging datasets such as Caltech [33] and ETH [37]. As opposed to the optimisation of a single pedestrian detector (e.g. *Roerei* [7]), we propose a different approach to increase the detection accuracy: combining pedestrian detectors.

Take for example the frame in figure 3.1a. Here, the detections of three pedestrian detectors, separately evaluated on the same image, are visualised. Note that none of them manages to find all pedestrians, due to the differences in features and training process. The result of the combination algorithm for these detectors, which we describe in this chapter, is shown in figure 3.1b. As we can see, most of the correct detections from the different detectors are kept, while incorrect detections are pruned.

We therefore address a fundamental question: how should we combine the detection results of multiple pedestrian detectors to allow for a higher accuracy rate? Seeking such a strong combination rule is not a trivial task, since many design choices are to be considered in this process. In this chapter we tackle these questions, and present a generic framework to achieve these pedestrian detector combinations. Traditionally, combining multiple pedestrian detectors is performed using a basic *AND* or *OR* rule. Later, in section 3.3.7, we propose another technique, where Non-Maximum-Suppression is performed on the detection results of all detectors together. Our framework utilises a more profound approach, and exploits information from each pedestrian detector. In a nutshell: our framework combines the detection scores from multiple



(a) The detectors used are *Ours-HOG* in blue, *Ours-DPM* in cyan and *Ours-ICF* in magenta.

(b) Result of our combination rule.

Figure 3.1: Left: An overview of the common object detection approach. Right: the result of our combination approach.

pedestrian detectors using a weighted sum to calculate the final detection result scores. For this we came up with an approach to measure certain intuitive properties from each detector to take into consideration, such as the *confidence* of a detector and the *complementarity* between detectors, in order to determine the optimal combination to further increase the accuracy. One could argue that, to determine the optimal weights, a machine learning strategy (e.g. Support Vector Machines) could be utilised. This is difficult since pedestrian detectors do not provide a detection probability score for all positions in the image. To achieve this, the pedestrian detection algorithms themselves should be adapted. However, since many pedestrian detection algorithms use completely different approaches, determining such a probability score in a fair way is an impossible task. Our approach allows combining the detection results from all detectors *out-of-the-box* and, as we show further on, still manages to find the optimal combination weights. Note that although we use our combination approach for the combination of pedestrian detectors, it should be applicable for object detection in general.

Our main contributions are two-fold:

- We propose a generic methodology that allows for an efficient combination of an arbitrary number of pedestrian detectors.
- We perform thorough experiments, and propose combinations that achieve better than state-of-the-art accuracy results.

Evidently the calculation time will increase when running multiple pedestrian detection techniques. A balance needs to be made between the accuracy and the computational complexity. In this chapter we will mainly focus on the accuracy. However, recent advances concerning speed improvements on existing pedestrian detectors show promising results. Fast multi-core implementations (GPGPU or multi-core CPUs) currently achieve reasonable to excellent processing speeds. In later chapters we will go deeper into the use of these approaches to improve detection speed of the pedestrian detector algorithms separately, which evidently will benefit the calculation time if used in a combination.

The remainder of this chapter is structured as follows. Section 3.2 describes related work on this topic. In section 3.3 we propose our combination approach and give detailed information on our combination parameters based on three of our detectors. The experiments we performed using all of our implementations are described in section 3.4. Finally, in section 3.5 we present our conclusions on this topic.

3.2 Related Work

Although integration strategies are applied in other research domains, to the best of our knowledge only few works concerning the optimal integration of multiple object detectors exist. Most work in which multiple object detectors are combined only use naive *AND* or *OR* combination rules. For example De Beugher et al. [20] proposed to use of the OR combination rules using the *Deformable Part Model* detector [40] for detecting the upper-body of pedestrians, and the face detector proposed by Viola and Jones [89], to improve the detection results. A more intelligent approach is given in [62], where the authors present a probabilistic framework in which they combine the detection score of an object detector (DPM) with Conditional Random Fields (CRF) to achieve better scene understanding. However, they only use a single object detector. In recent work, Mathias et al. [65] uses multiple object models, based on the same *SquaresChnFtrs* detector [7], to cope with occlusion (new models are trained for a specific occlusion level). The different detection results are then combined in a weighted sum into a single detection score (with the area of each model taken into account).

Our work significantly differs from all of the previously mentioned works. We propose an approach that aims to combine the detection results of multiple independent object (pedestrian) detectors in the most optimal manner. In essence, our goal is to determine the best possible combination rule to maximally increase the accuracy. We utilise specific information from each individual

detector to obtain such an optimal combination rule. Furthermore our framework allows the combination of an arbitrary number of detectors. As shown in section 3.4, the accuracy of an optimal combination outperforms existing state-of-the-art pedestrian detectors. In the next section we propose our approach and motivate our design choices.

Shortly after we published "The Combinator: optimal combination of multiple pedestrian detectors" [26], Xu et al. presented a very similar work in [92]. Although their work is written from the point of view of information fusion, their approach has a lot of similarities with our approach. They use the same Matlab based evaluation framework as a starting point [33], and just like our work they take a confidence measure and complementarity measure into consideration to determine the final detection score. On the Caltech dataset, our approach turns out to be slightly more accurate when combining HOG, ICF and DPM (which is the final combination we proposed in [26]).

3.3 Approach

As mentioned above, currently only naive pedestrian detector combinations are used. Here we try to increase the accuracy by combining the detection results of multiple pedestrian detectors in a more profound approach. That is, we propose to combine the detection scores from each individual pedestrian detector using a weighted sum. Our goal is then to find these optimal combination weights, such that they exploit the strengths of each individual detector. For example, the DPM detector has excellent accuracy for high- to medium-resolution pedestrians, for which the accuracy of the HOG detector is lower; however the DPM detectors requires to find details which are not available in pedestrians in low resolution, such that rigid template models such as HOG are more effective. The work of Park et al. [73] is based on this property to improve accuracy for these specific pedestrian detectors. Note here that the focus of their work lies in an improvement due to resolution difference, and will therefore be compatible with our approach.

The combination approach we are proposing tries to find an optimal manner to combine the detection scores of detectors when they detect at the same location. Therefore the first step is to determine which detections belong together and are to be combined. This is explained in more detail in section 3.3.1. The detections in each cluster are combined in a weighted sum. Our weighted sum approach uses, next to a normalised detection score, two measures: a confidence value and a complementarity value, which will be discussed in more detail in section 3.3.3 and 3.3.4 respectively. To determine these parameters for each

detector, we have to determine an operating point at which the detectors can be treated equally. How we determine this point is discussed in section 3.3.2. Section 3.3.5 describes how the final score for each of the found clusters is determined. Finally, this section concludes with a validation of our approach based on combining two detectors, showing that it manages to reach an optimal accuracy.

3.3.1 Clustering detections

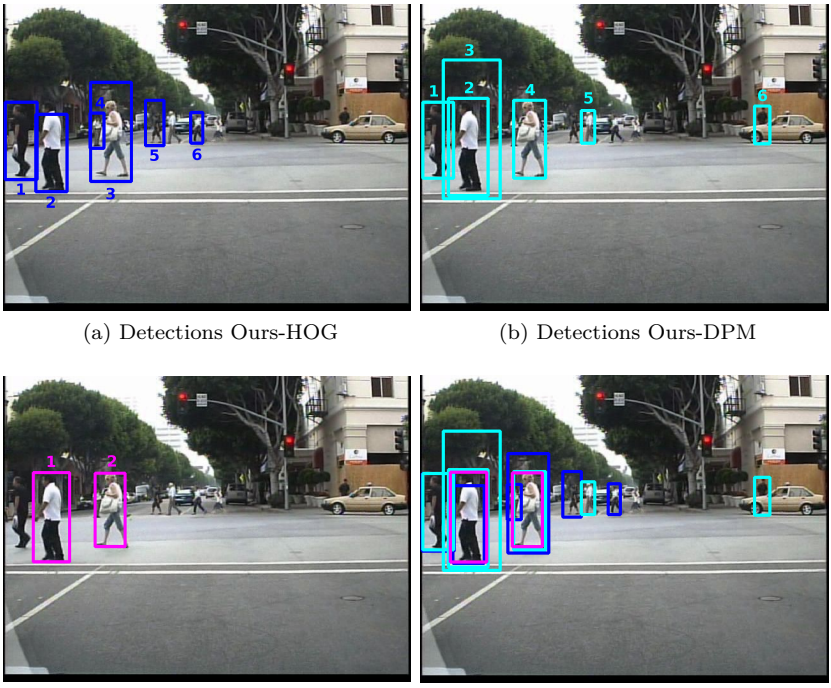
An important task when combining detections is to determine which detections need to be clustered together, to be considered as belonging to the same detections and should be combined. As a guidance rule, we use a 50% overlap criterion for pair-wise clustering, which is defined by equation 3.1 (the intersection between the detection boxes need to be at least 50% of each detection). We define this as a two-way commutative operation, such that $Overlap(A, B)$ is the same as $Overlap(B, A)$. Since we only want to consider combinations between different detectors, the first step we take is running a Non-Maximum-Suppression algorithm for each detector separately.

$$\frac{area(BB_1 \cap BB_2)}{max(area(BB_1), area(BB_2))} \geq 0.5 \tag{3.1}$$

The clustering problem can be solved by representing the detections as a graph, where each detection is represented as a node, and a sufficient overlap (based on the 50% overlap-criterion) is represented as an edge between the nodes. Figure 3.2 shows an example frame with the detections of *Ours-HOG*, *Ours-DPM* and *Ours-ICF*. Based on the 50% overlap-criterion, we create the corresponding graph, which is shown in figure 3.3.

The clustering problem can now be defined as determining all the groups where each node is connected with each node by an edge. Such a group is in graph theory known as a *clique*. In 1973 C. Bron and J. Kerbosch [16] proposed an algorithm that efficiently solves this problem. According to [83], this algorithm turns out to still be one of the fastest to solve this particular problem (on static graphs). For our experiments, we use the implementation of this algorithm available in the C++ graph library of Boost [82]. Detections that do not have a sufficient overlap with other detections, and so are single, are added as an independent group. For our example image, we end up with nine groups:

- Two groups where the detection is covered by all three detectors
- One group where the detection is covered by Ours-HOG and Ours-DPM



(a) Detections Ours-HOG

(b) Detections Ours-DPM

(c) Detections Ours-ICF

(d) Detections all detectors

Figure 3.2: Detections on an example frame

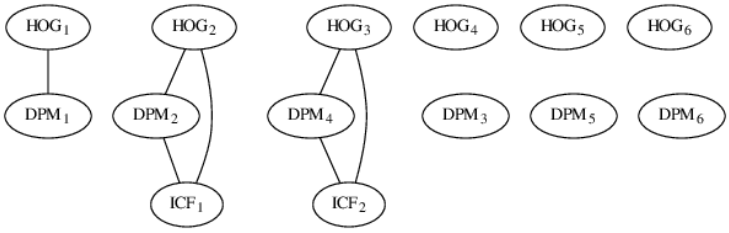
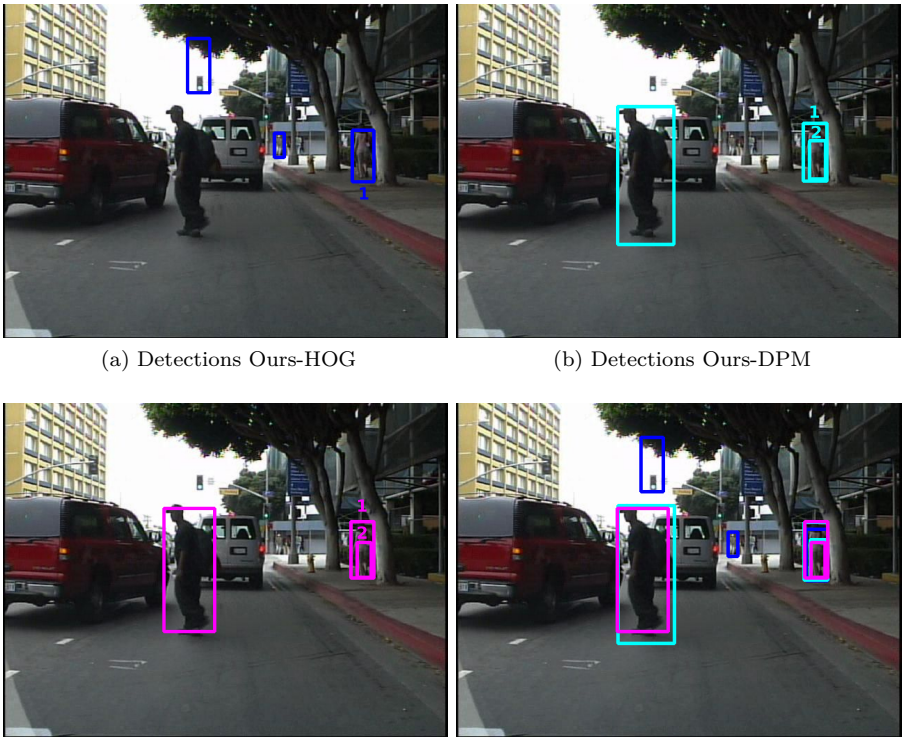


Figure 3.3: Graph resulting from the detections on figure 3.2.



(a) Detections Ours-HOG

(b) Detections Ours-DPM

(c) Detections Ours-ICF

(d) Detections all detectors

Figure 3.4: Visualisation of the detections on an example frame. Note the double detections on the rightmost pedestrian.

- Three groups where the detection is only covered by Ours-HOG
- Three groups where the detection is only covered by Ours-DPM

Note that a detection can be part of multiple groups. An example of this can be found in figure 3.4, where the rightmost pedestrian is detected multiple times (even by the same detector). In figure 3.5 the graph-representation of the detections on this specific pedestrian are shown. The issue that arises here, is that the smallest detection of DPM (DPM_2) has sufficient overlap with the HOG (HOG_1) detection, but only 46% with the largest ICF (ICF_1) detection. Next to that, the smallest ICF (ICF_2) has only 44% overlap with the HOG (HOG_1) detection.

Due to this, we end up with three groups for the same "object instance":

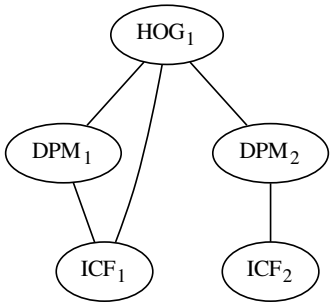


Figure 3.5: Subgraph of rightmost detection in figure 3.4

- One group with all detectors (DPM_1, HOG_1 and ICF_1)
- One group with DPM_2 and HOG_1
- One group with DPM_2 and ICF_2

The detections in each group will be handled as a combination according to the technique we describe in the remainder of this chapter.

3.3.2 Determining combination parameters

The challenge of combining the detections in each cluster lies now in how much we *trust* each detector as part of a combination. Therefore, we propose the use of two measures: *confidence* and *complementarity*.

- **Confidence**: The confidence value indicates how *good* a detector performs. It gives an indication about the probability of a detection by detector i being a correct detection (further indicated as $c_{\text{conf}(i)}$). This value is calculated independently from the other detectors in the combination.
- **Complementarity**: Each pedestrian detector uses a specific design methodology (e.g. different feature pools or classifiers). This measure tries to indicate how *different* the detectors are (further referred to as $c_{\text{compl}(i)}$), and as such how much additional information is obtained when multiple detectors detect a pedestrian at the same location. This value is not fixed per detector, but depends also on the other detectors in the combination.

	Ours-HOG	**Ours-ICF**	**Ours-DPM**
Confidence coefficient	0.102	0.137	0.149

Table 3.1: Confidence coefficients for our three example detectors.

We use these two measurements in a weighted sum to combine the detection scores from two or more pedestrian detectors. Below we explain in detail how each of these measurements is determined.

Since each detector has specific tuning parameters (e.g. thresholds and Non-Maximum-Suppression) we need to determine an unbiased way to compute the confidence and complementarity coefficients over all detectors. To achieve this, we select a fair operating point (or detection threshold) on the Precision-Recall (PR) curve for each detector. More information about the PR curve is given in section 2.4.

We determine the optimal threshold using an *equal number of detection windows*. By choosing the threshold of a detector strict or sloppy, the number of detections returned from a test dataset becomes smaller, respectively larger. To find a fair operating point across different detectors, we choose a threshold for each detector such that the number of returned detection windows, N, is equal for these different detectors on the same dataset. Here N is a fixed value for all detectors, determined as a percentage of the number of ground truth detections on our calibration dataset.

This is visualised in figure 3.6 for three pedestrian detectors: *Ours-HOG*, *Ours-ICF* and *Ours-DPM* where the markers indicate the optimal threshold operation point (N is chosen as 50% of the number of ground truth detections). During this chapter we will illustrate our combination approach using these three algorithms. In section 3.4 we will apply our combination approach on all detectors we described in section 2.5 for all possible combinations of two and three detectors.

In the next subsections we give an overview of how each of the coefficients defined above are determined. Note that all further calculations in the subsections below use this optimal threshold, thus ignoring detections with a lower detection score.

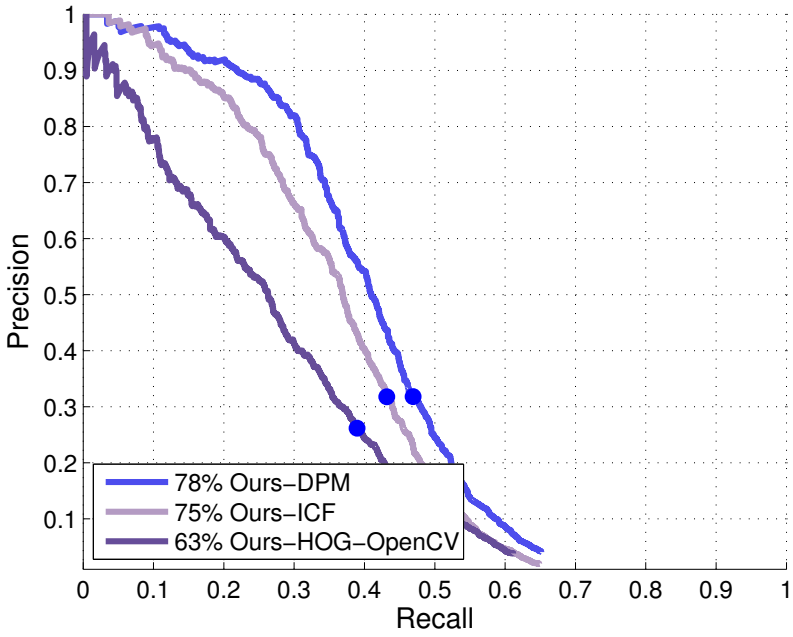


Figure 3.6: The optimal thresholds (PR operating points) by using a fixed number of detection windows ($N = 50\%$).

3.3.3 Confidence coefficient

The confidence coefficient $c_{\text{conf(i)}}$ gives an indication of the detection accuracy of detector i. This information is independent of other detectors in the framework, and is based on the accuracy that a detector achieves on a specific dataset.

Several statistics can be used for this measure, e.g. the average precision (AP). We propose to use the area from the rectangle through the origin and the operating point on the PR curve for each detector i (the product of the Precision and Recall value at the operation point), since this emphasizes the importance that both the *Precision* and the *Recall* should be high. Table 3.1 gives the confidence coefficients for our example detectors.

	Ours-HOG	**Ours-ICF**	**Ours-DPM**
Ours-HOG	0	0.377	0.393
Ours-ICF	0.341	0	0.340
Ours-DPM	0.331	0.315	0

Table 3.2: Complementarity matrix for three detectors

3.3.4 Complementarity coefficient

Different detectors use different design methodologies and feature pools. Therefore, each pedestrian detector reacts differently to a specific image patch. Combining pedestrian detectors that are complementary w.r.t. each other thus could yield better detection results. Our complementarity coefficient $c_{\text{compl(i)}}$ tries to indicate how *different* these pedestrian detectors react, and thus what the additional value is when they find a detections at the same location. When multiple detectors with very different detection approaches yield a detection at the same image location, the chance of that being a true detection increases significantly (much more than when e.g. multiple detections from rather redundant pedestrian detectors using the same approach are found).

As an example, the frame in figure 3.1a visualises three detector outputs (see caption for colour coding). These detectors give significantly different detections. For example only Ours-HOG manages to find the small pedestrian in the centre, while Ours-DPM is able to detect the occluded pedestrian at the right. Some locations are covered by more than one detector, indicating a higher probability that these are correct detections. If combined efficiently, an optimal accuracy is achieved.

To determine a complementarity coefficient for each detector, we first calculate the pairwise complementarity score $w_{i,j}$ between two detectors i and j. This is done as follows. We compare the detection performance over the Caltech training dataset. For each frame, each detector is pairwisely compared. The number of detections from a specific detector, which are not covered by the other detector, according to equation 3.1, is determined.

These are then summed over all frames, and divided by the total number of detections for that detector. For example (on one frame), in figure 3.1a *Ours-HOG* has six detections of which three are covered by *Ours-DPM* (which leaves three), thus $w_{Ours-HOG,Ours-DPM} = 50\%$. *Ours-ICF* covers two of the six detection from *Ours-DPM* resulting in $w_{Ours-DPM,Ours-ICF} = 66.7\%$. If done for each detector pair, this results in a square complementarity matrix,

visualised for three detectors, on the whole training set, in table 3.2. Note that no annotation data is used; the fact that a detection is correct or not is irrelevant for the complementarity coefficient. This information is already included in our confidence coefficient. The complementarity coefficients aims to indicate how much extra information each specific detector introduces in the case of overlapping detections.

During the combination of the detection results, this complementarity matrix is used to calculate the complementarity coefficients of overlapping detections as follows. For overlapping detections, we first extract the corresponding square sub-matrix (containing only the *relevant* detectors - those that account for one of the detections) from the total complementarity matrix. Next we calculate a single *average complementarity coefficient* C_i for each detector i involved in this overlapping detection, using the individual pairwise complementarity scores $w_{i,j}$:

$$C_i = \frac{\sum_{j=1}^{n} w_{i,j}}{n-1} \tag{3.2}$$

Where n indicates the number of relevant detectors involved in the overlap. Note however that the final complementarity value used for each detector depends on the other detectors in the combination.

The function used to calculate the final complementarity value of a detector should have the following properties:

1. When a detector in the combination is completely complementary ($C_i = 1$) it yields only valuable information and thus we set $c_{\mathrm{compl}(i)} = 1$, independent of the other detector(s).
2. When all detectors are equally complementary ($C_A = C_B$) they are given the same score $c_{\mathrm{compl}(A)} = c_{\mathrm{compl}(B)}$.
3. In the extreme case where all detectors are completely redundant ($C_A = 0, C_B = 0$), which is the case when the same detector is used multiple times, this score is equally divided between the detectors $c_{\mathrm{compl}(A)} = c_{\mathrm{compl}(B)} = 1/2$ (in the case of two detectors).

A function that follows these constraints for an arbitrary number of detectors is:

$$c_{\mathrm{compl}(i)} = \frac{1}{n}\left[1 + \sum_{\substack{j=1 \\ j\neq i}}^{n}(C_i - C_j) + \sum_{\substack{j=1 \\ j\neq i}}^{n}(C_i C_j) + (\sum_{j=1}^{n} C_j - 1)(1 - C_i) + \prod_{j=1}^{n}(1 - C_j)\right] \tag{3.3}$$

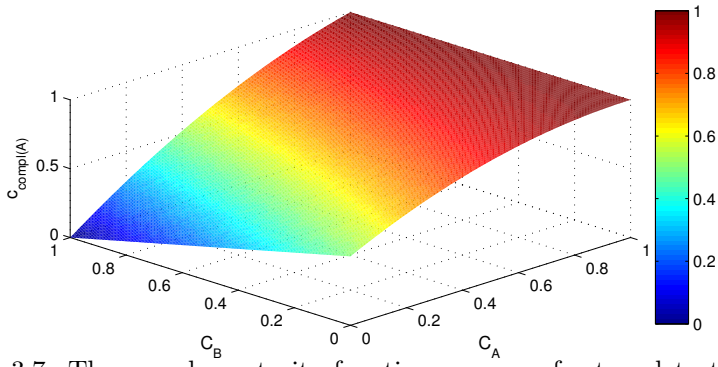


Figure 3.7: The complementarity function $c_{\text{compl}(A)}$ for two detectors. This function complies with the properties we want the complementarity value to have (explained in the text).

Where C_x is the individual average complementarity coefficients and n indicates the number of relevant detectors involved in the overlap. These n complementarity coefficients thus summarise how each individual detector involved in this overlap should be weighted as such to maximally exploit the information potential of each specific detector.

When pairwise complementarity is used, this formula is reduced to

$$c_{\text{compl}(A)} = \frac{1}{2}\left[1 + (C_A - C_B) + C_A C_B + C_A(1 - C_A)\right] \tag{3.4}$$

For clarification, the complementarity function $c_{\text{compl}(A)}$ for $n = 2$ is visualised in figure 3.7.

To illustrate, suppose that an overlapping detection with both *Ours-HOG* and *Ours-ICF* is found ($n = 2$). First, we extract the 2×2 complementarity sub-matrix. Next, using eq. 3.2, we determine the average complementarity coefficient for each detector ($C_{Ours-HOG} = 0.377, C_{Ours-ICF} = 0.340$). Finally, we calculate the complementarity coefficient for each detector using eq. 3.4, yielding $c_{compl(Ours-HOG)} = 0.71$ and $c_{compl(Ours-ICF)} = 0.6515$.

Since both the confidence values and the complementarity values for all detectors in a combination are determined, we can now determine the final detections score.

3.3.5 Combining detection results

Using the coefficients determined as described in section 3.3.3 and 3.3.4, the actual combination for each group after clustering (section 3.3.1) can be performed. The final detection score is determined based on the (normalised) output scores S_i of the n overlapping detections, the confidence coefficient and the complementarity coefficient of each detector i that yielded a detection there, calculated as mentioned above, using a weighted sum:

$$S_{\text{final}} = \sum_{i=1}^{n} c_{\text{conf}(i)} c_{\text{compl}(i)} S_i \tag{3.5}$$

Also here we use the multiplication of the two parameters since the best combination will be performed when both the confidence of each detector and the complementarity between the detectors are high.

For the final bounding box we return the average (using the individual detection scores) over the overlapping bounding boxes. For all non-overlapping detections we multiply the detection score with the confidence value of that detector and the complementarity coefficient, calculated as if this detection overlapped with all detectors. Our confidence and complementary coefficients are chosen in such a way that multiple detections from complementary detectors with high confidence return high scores, whereas redundant detectors with low confidence evidently output lower total detection scores. We coined the use of this combination approach *The Combinator*. In section 3.4 our experiments show that our approach achieves very good accuracy results.

3.3.6 Validation of our approach

In this section we validate the combination rule of *The Combinator* (equation 3.7), and show that our weighted sum approach using the confidence coefficient and complementarity coefficient as defined above, reaches the optimal solution (the combination with the lowest miss rate) on the Caltech test set. This is done as follows. Take for example the combination of two detectors A and B. Our combination rule then becomes:

$$S_{\text{final}} = c_{\text{conf}(A)} c_{\text{compl}(A)} S_A + c_{\text{conf}(B)} c_{\text{compl}(B)} S_B \tag{3.6}$$

Since only the relative weights are important, this can be reformulated as:

$$S'_{\text{final}} = \alpha S_A + (1 - \alpha) S_B \tag{3.7}$$

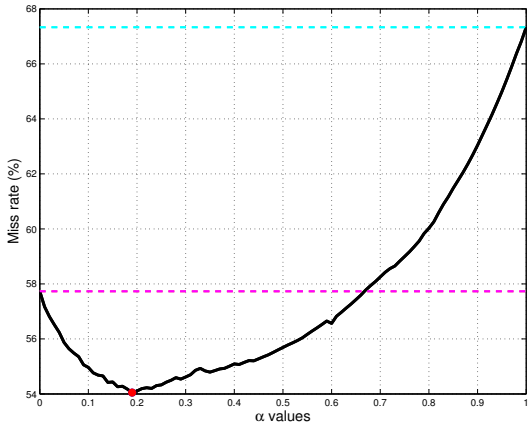


Figure 3.8: The average miss rate (%) in function of the α value. Dotted lines indicate the accuracy of both individual detectors (*Ours-HOG* at the top, *Ours-ICF* at the bottom). The dot is the combination rule we propose, derived independently from this experiment.

Thus, if we let α vary from zero to one, all possible relative combinations are evaluated.

We calculate for each of these combinations the miss rate of the resulting combined detector on the test set. Figure 3.8 gives these results for the combination of *Ours-HOG* and *Ours-ICF*. At the extreme values 0 and 1 of α, the accuracy score equals that of both individual detectors (indicated with the dotted lines). An optimal combined accuracy is reached for a specific value of α between these two boundaries. The dot indicates the value of α calculated using our combination rule.

As can be seen, our proposed combination rule manages to find the optimal combination weights. Note that our weights are calculated based only on the confidence and complementarity measures. These are easily extracted from the detection results, thus avoiding the need to perform an exhaustive search over all possible combinations like we do in this validation experiment.

Note that this validation is obtained using our Matlab implementation of the original publication [26], while our experiments of section 3.4 are performed on our more accurate C++ implementation of [21], which imposes an additional accuracy improvement.

3.3.7 Pool combination

Next to the combination rule of section 3.3.5 (equation 3.7), we proposed an alternative combination approach in [21], which we coined *Pool combination*. The used approach is very similar to that used to combine mixture models in the *DPM* detector [39]. Since the goal of normalising the detection scores is to obtain detection scores in the same range as other detectors, we can compare them fairly. Instead of creating a cluster of overlapping detections, we use an easier approach and perform Non-Maximum-Suppression over the detections of all detectors together. The difference between this approach and an OR-combination is avoiding the requirement of clustering the detections. Keep in mind, though, that the score normalisation does not take into account any information of the detectors. In section 3.4 we compare the use of our optimised combination (*The Combinator*) method and this simplified strategy.

3.4 Experiments and Results

To illustrate the potential of our combination approach, we performed thorough accuracy experiments. For our experiments we use the detection results of our detector implementations, described in section 2.5, evaluated on the Caltech dataset [33]. This dataset consists of about 250,000 frames of which each 30th frame is used for evaluation (resulting in about 8300 frames). All combination coefficients as mentioned above were first calculated on the training set (set00 - set05, 4250 frames). As optimal operating point we use $N = 50\%$ of the number of ground truth detections. Next, our combination rule was executed on the test set (set06 - set10, 4024 frames), using the *reasonable* settings. As the experiments indicate, a combination of detectors can lead to high accuracy improvements, and specific combinations achieve better than state-of-the-art detection results.

Figures 3.9 and 3.10 show the results of the *Pool combination* approach, as described in 3.3.7. The accuracy results we obtain using our best performing combination approach, *The Combinator* approach as described in section 3.3.5, are shown in figure 3.11 and figure 3.12. The results are divided into combinations of two detectors and three detectors respectively. Table 3.3 and 3.4 give the comparison of these two approaches on the 0.1 FPPI point for the combinations. Based on this, we can point out that our *The Combinator* approach decreases the *Miss Rate* on average by 4.5% over *Pool combination*. This is the gain we obtain by taking the difference between detectors into account, instead of only the normalised detection scores.

Note that, except for the combinations *DPM-ACF-SqrtICF* and *DPM-ICF-ACF*, adding more detection algorithms to a combination with *The Combinator* always improves accuracy. Although this seems evident, since more information is taken into consideration, it is remarkable since we only use two values to describe the complete behaviour of a detection algorithm. But as we noted in the related work section (3.2), the similar approach described in [92] uses a more dynamic confidence measure, and does not obtain a higher accuracy.

When combining, it is important that both the confidence and the complementarity are high. When we look into the accuracy of combining the *Channel Based* detectors (*ICF, ACF and SqrtICF*), we see that the accuracy gain is rather limited, since they use the same feature channels to work with, and thus have a low complementarity. Note that, although ACF and SqrtICF are more accurate than ICF, combining each of them with ICF is more accurate than combining them together, since the complementarity with the ICF detector is larger. Knowing this, it is no surprise that the combination between the three *channel based* detectors is the least accurate combination of three, while the other combinations contain detectors with very different training methodology and features.

As a rule of thumb, we can state that the best combination is made when both complementarity between the detectors and confidence of the detectors are high, e.g. accurate detectors from different detector families, such that the chance that the missed detections of one detector are covered by the other(s) increases. Note that the foundation of our combination approach (equation 3.7) reflects exactly our rule of thumb, which is supported by the accuracy results of our experiments.

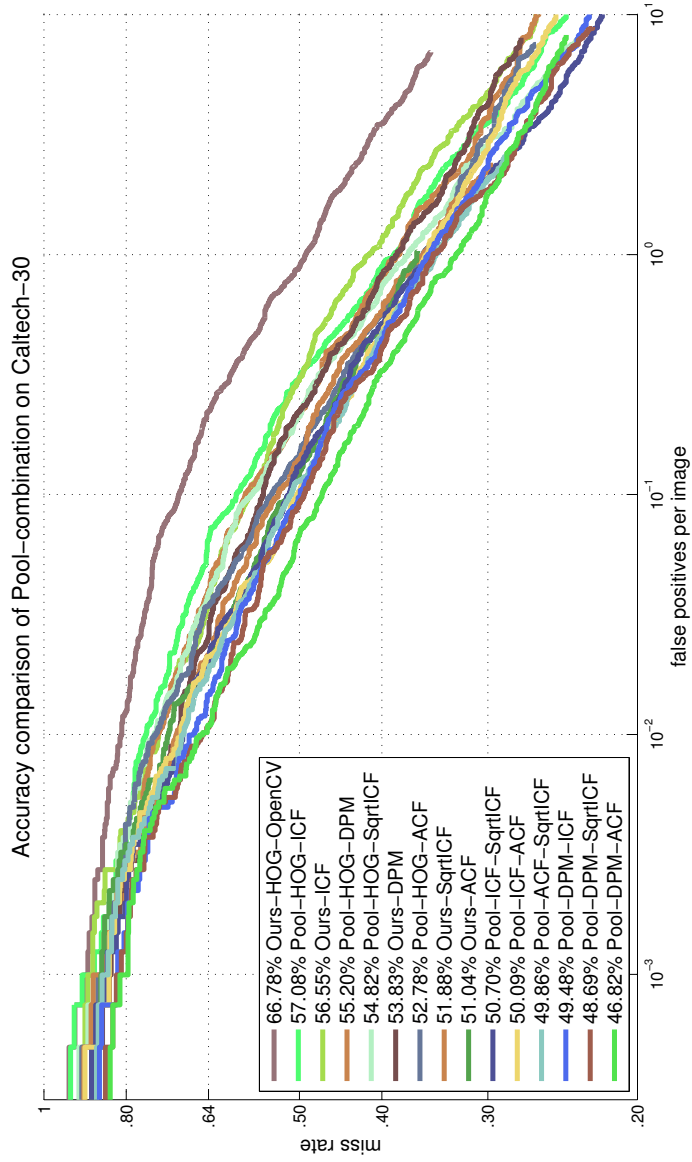


Figure 3.9: Accuracy of pair-wise combinations using *Pool-combination*.

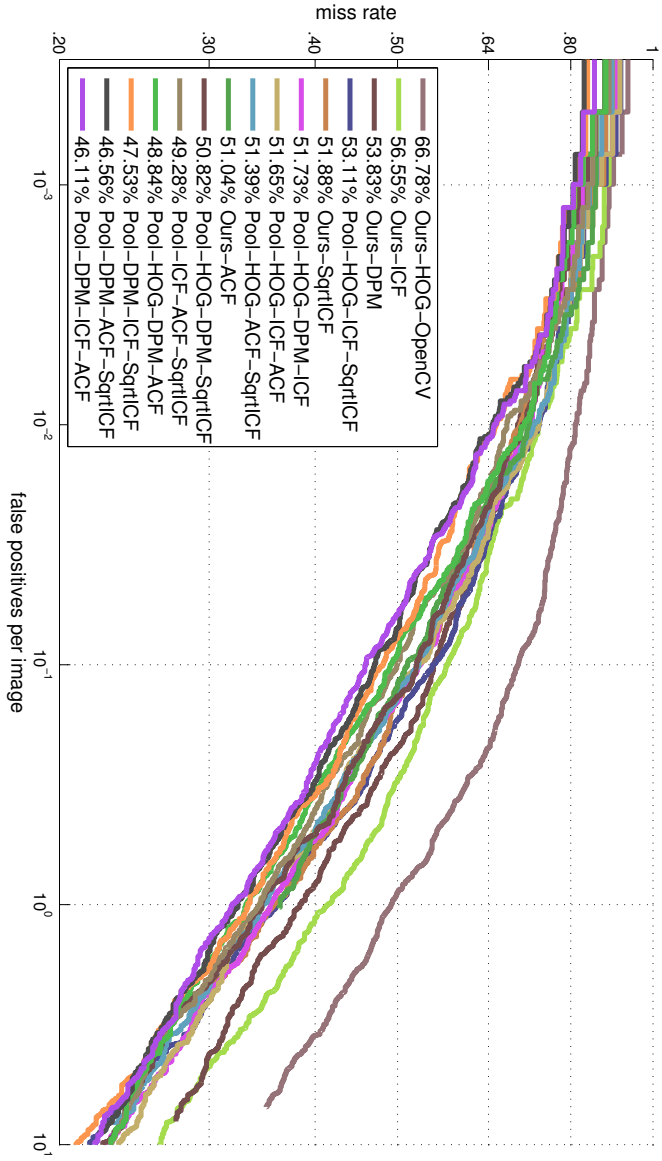


Figure 3.10: Accuracy of combinations of three detectors using *Pool-combination*.

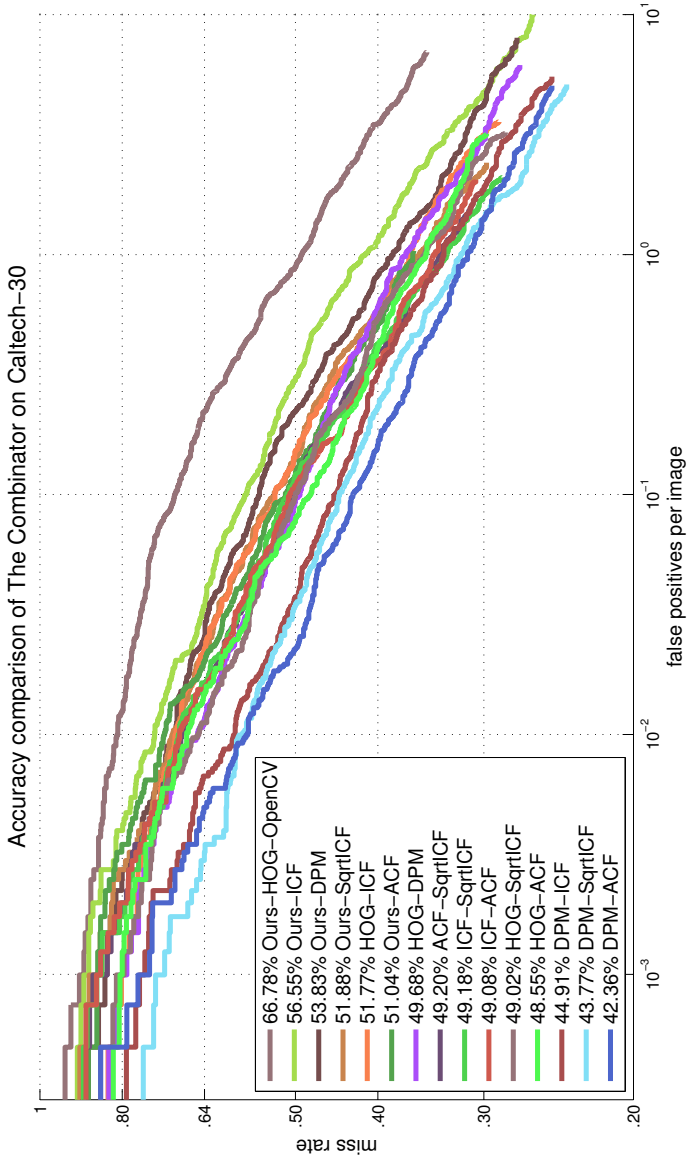


Figure 3.11: Accuracy of pair-wise combinations using *The Combinator*.

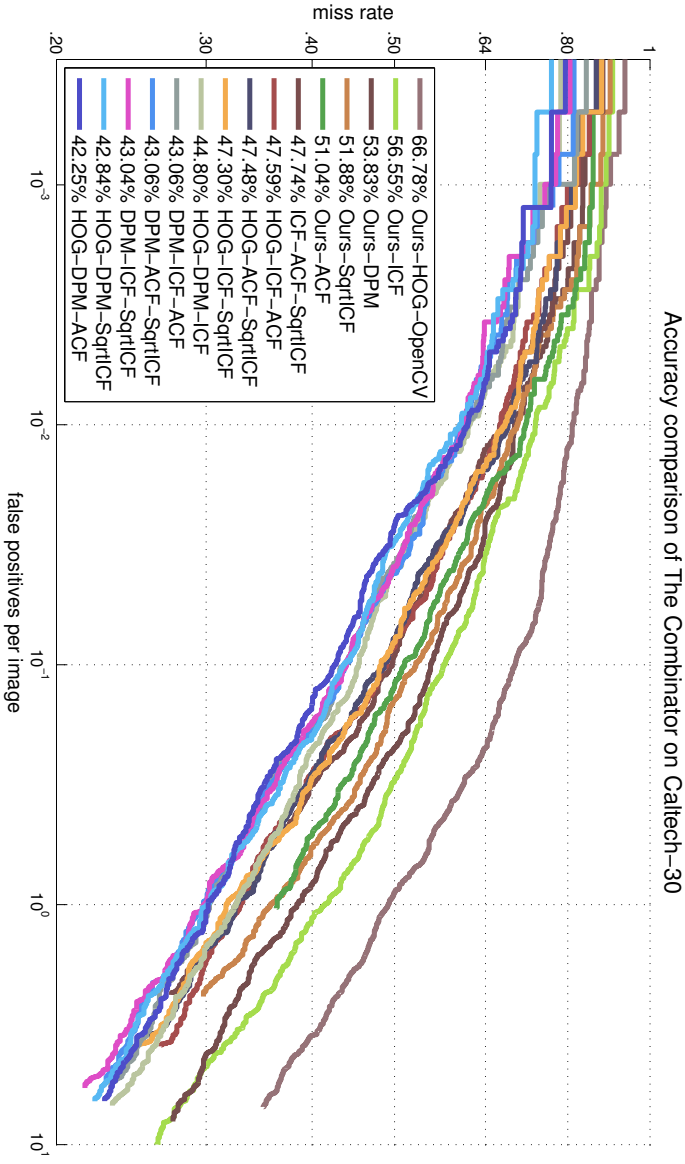


Figure 3.12: Accuracy of combinations of three detectors using *The Combinator*.

	Pool Combination	The Combinator
HOG-DPM	0.58	0.49
HOG-ICF	0.59	0.53
HOG-ACF	0.54	0.48
HOG-SqrtICF	0.57	0.50
DPM-ICF	0.50	0.45
DPM-ACF	0.47	0.43
DPM-SqrtICF	0.49	0.44
ICF-ACF	0.50	0.50
ICF-SqrtICF	0.51	0.51
ACF-SqrtICF	0.51	0.50

Table 3.3: Resulting miss rate of combination approaches using two detectors on 0.1 FPPI.

	Pool Combination	The Combinator
HOG-DPM-ICF	0.53	0.45
HOG-DPM-ACF	0.49	0.43
HOG-DPM-SqrtICF	0.53	0.44
HOG-ICF-ACF	0.53	0.49
HOG-ICF-SqrtICF	0.55	0.48
HOG-ACF-SqrtICF	0.53	0.49
DPM-ICF-ACF	0.46	0.43
DPM-ICF-SqrtICF	0.46	0.44
DPM-ACF-SqrtICF	0.48	0.44
ICF-ACF-SqrtICF	0.50	0.50

Table 3.4: Resulting miss rate of combination approaches using three detectors on 0.1 FPPI.

Besides our approach, for each experiment we also performed the *AND* (only keep overlapping detections) and *OR* (keep all detections) combinations. Figure 3.13 displays the accuracy results of our combination rule for *Ours-ICF + Ours-DPM*, compared with the AND and OR combination of the same detectors. Our proposed combination approach easily outperforms these naive combination rules. A similar trend is noticed for our other combinations versus the AND and OR combination rules.

Finally, figure 3.14 displays the accuracy of our top performing combination (*Ours-HOG + Ours-DPM + Ours-ACF*) compared to the state-of-the-art. Although recent publications [94, 8, 98] did obtain higher accuracy on the Caltech test set, these detectors are trained on the Caltech training set which

	DPM-ACF	HOG-DPM-ACF
HOG	/	40.41%
DPM	95.24%	56.77%
ACF	4.69%	2.77%
Overhead	0.06%	0.04%

Table 3.5: Deviation of the time spent in each part while applying *The Combinator*.

imposes an advantage. The only information we use here is how well each of the algorithms used in the combination performs on the trainingset, in the belief we can expect the same behaviour on the testset. The most accurate result of an INRIA-trained model to our knowledge is the *Roerei* detector described in [7] which is also shown on the curve. As we can see, our combination turns out to be more accurate under 2 FPPI (which equals 8% precision), and for most applications the accuracy should be drastically higher.

Further note the reduction from 0.1 FPPI to 0.023 FPPI at the same miss rate when comparing our combination with *Ours-ACF*, the best performing algorithm in the combination.

The division of time consumed in the different parts of *The Combinator* for our best performing combinations, is shown in table 3.5. As we can see forms the evaluation of DPM and HOG the major shares of the evaluation time, which means that improving the speed of these will have the largest impact on the resulting evaluation time. In chapter 4 and 5 we discuss techniques that can be used for this purpose. The time spent at the clustering of the detections is for the *DPM-ACF* combination approximately 24μs while for *HOG-DPM-ACF* 46μs. Note however that measuring such small times is strongly influenced by measurement errors. Although we see that the time spent clustering the detection becomes almost twice as large by adding an additional detector, the influence is negligible in the final evaluation time. The processing times we reach are 0.306 fps and 0.176 fps for *DPM-ACF* and *HOG-DPM-ACF* respectively. Important to mention is that the accuracy improvement of adding the HOG-detector to the *DPM-ACF* combination is small, while the speed drops significantly. Therefore we recommend to use just the combination *DPM-ACF* instead.

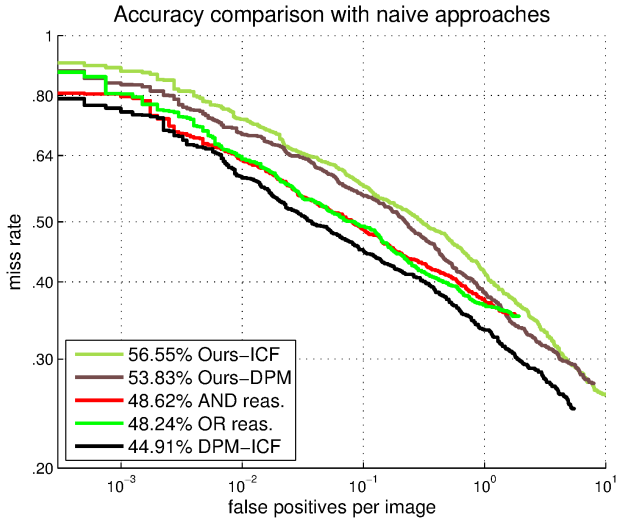


Figure 3.13: Precision-Recall curve of our combination approach for *Ours-ICF + Ours-DPM*, compared with the standard *AND* and *OR* combinations.

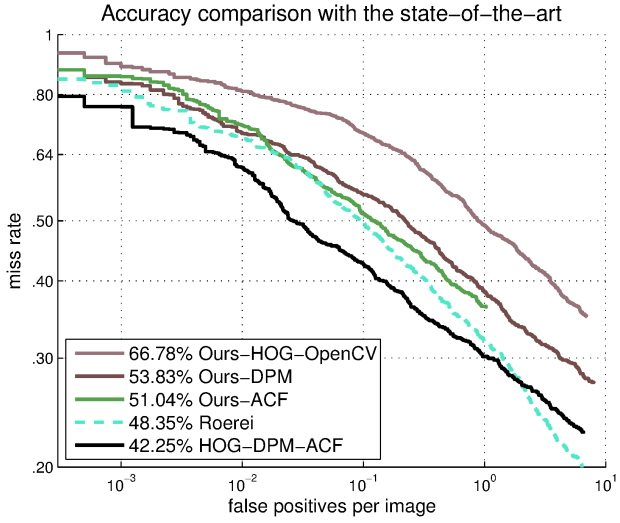


Figure 3.14: Accuracy comparison between our best combination and the state-of-the-art.

3.5 Conclusion

In this chapter we presented a generic pedestrian detector combination methodology to further increase the detection accuracy. Our approach allows for the combination of an arbitrary number of pedestrian detectors, and manages to achieve an optimal combination rule. For this we introduced two measures: *confidence* and *complementary*. Using these measures the detection scores of multiple pedestrian detectors are combined in a final detection score.

Using this approach, we performed a large amount of experiments using the five detection algorithms we presented in section 2.5. We compared the results of this approach with the traditionally used (naive) *AND* and *OR* approaches, our own *Pool combination* approach, and the state-of-the-art. Hereby we point out the high accuracy improvement that can be achieved when using pedestrian detection algorithms in a combination.

In this chapter we focussed on the improvement of the detection accuracy, while ignoring the impact on detection speed. In chapter 4 and 5 we will focus on improving the detection speed of the algorithms separately, which evidently improves the detection speed when used in a combination.

Chapter 4

Implementation Improvement

In chapter 2 we presented multiple detectors, each designed to detect pedestrians in images. In chapter 3 we used these to improve the detection accuracy by combining the results of these detectors using *The Combinator* approach, which takes the *Confidence* of each detector and the *Complementarity* between detectors into account.

For real-life applications, it is required that these pedestrian detection algorithms run as fast as possible, while at the same time keeping their accuracy at top level. There are essentially three methods to improve the detection speed. The first is altering the algorithm itself, such as the integration of a cascade-based evaluation [41, 12, 89] instead of evaluating the whole model at each location. A second, generally applied, method is the reduction of the search space of the detector by integrating scene knowledge [18, 6], which we discuss in chapter 5. The third method, which we discuss in this chapter, focuses on exploiting hardware capabilities to improve detection speed, more specifically using parallelisation. Note however that the techniques we discuss in this chapter are complementary with the techniques we describe in chapter 5.

4.1 Introduction

In this chapter, we look into multiple approaches to improve detection speed of pedestrian detection algorithms by exploiting the use of parallelisation. We will describe multiple approaches to integrate parallelisation into our pedestrian detection algorithms, each having both advantages and disadvantages. In section

4.2 we give a brief introduction to parallelisation, followed by the related work on this topic in section 4.3. In section 4.4 we improve the detection time for frame-by-frame processing, by evaluating the layers of the feature pyramid in parallel. Hereby we reduce the latency between the moment of receiving the frame and the moment the detections are available. In section 4.5 we evaluate multiple frames in parallel, which helps to improve the throughput. In section 4.6, we describe the parallelisation approach we used in our publications "Pedestrian detection at Warp speed: exceeding 500 detections per second" [25] and "On-board real-time tracking of pedestrians on a UAV" [22], where the task of the application is divided into subtasks, each evaluated in a separate thread. The data is passed between threads by using queues. In section 4.7 we exploit the data-parallelisation opportunities for the feature pyramid of the *Deformable Part Model* detector, by integrating it on a *Graphical Processing Unit* (GPU). Finally in section 4.8 we give the conclusions of this chapter.

4.2 What is parallelisation?

Traditionally, most software was written for single core architecture. Herein, running the software was executing the instructions of the software sequentially. To obtain a higher processing speed without altering the software, a processing platform with a higher clock rate had to be used, such that more instructions could be executed in the same time span (or improve caching, branch prediction, instruction parallelisation such as SSE,...). Although only a single instruction could be executed at the same time, most operating systems supported the use of multi-tasking, by switching between tasks over time (time scheduling). This gives the illusion that multiple tasks actually ran at the same time on the CPU.

In the last decade, instead of increasing the clock frequency of the CPUs, the number of cores in a CPU was increased, with power efficiency in mind. According to [17], the power of a micro controller is calculated using the following equation:

$$P = CV^2 f \tag{4.1}$$

where C is the *capacitance*, V is the *voltage* and f is the *clock frequency* of the micro controller. Figure 4.1 visualises the dual core equivalent (same number of instruction can be executed) of a single core processor. In the dual core design, the frequency can evidently be divided by two, which also allows to lower the voltage which is directly related with the clock frequency. The capacitance however will increase with the amount of transistors in the micro controller, both to create an extra core and to communicate between the cores. The values for the voltage and capacitance in this example are based on [17] and

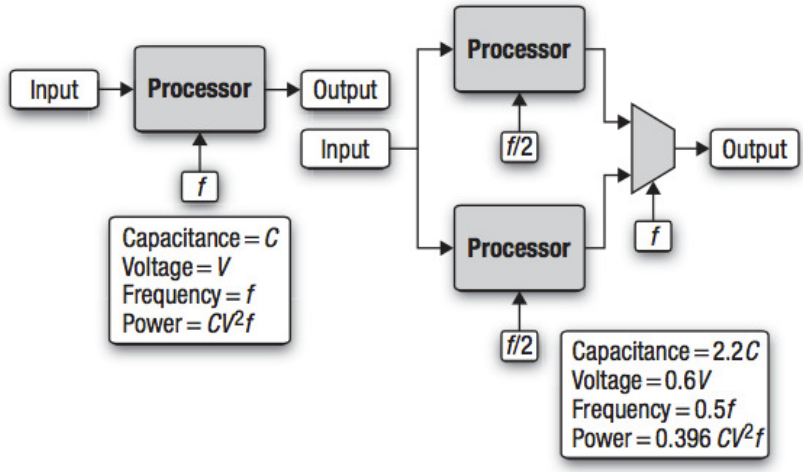


Figure 4.1: From a single core architecture to a dual core architecture with the same instruction throughput, taken from [66] and based on [17].

are measured on an actual chip. Note that these values are dependent on the components used in this example, but it clearly shows the benefit of a multi-core architecture based on power efficiency.

So now multiple tasks could actually run at the same time. Evidently, software needs to be altered to benefit from this, and the manner how parallelisation is used will determine how the performance improves.

A software program can be divided in a parallel part, in which the instructions do not depend on each other, and so could be evaluated in parallel, and a sequential part that is undividable, and thus has to run sequentially. Amdahl's Law gives us a theoretical estimate of the improvement that can be obtained by using parallelisation:

$$S = \frac{1}{(1-P) + \frac{P}{N}} \tag{4.2}$$

Herein P represents the share of the software that can run in parallel, so $(1-P)$ represents the sequential part, N stands for the amount of parallel instances and S the reduction in processing time of the algorithm. Note that this reduction is very optimistic, because it does not include overhead due to communication and synchronisation between tasks, constructing and destroying parallel tasks, ...

To execute tasks in parallel, there are mainly two possibilities. The first is using parallel processes. When this is used, an exact copy of the current process is created, including all the variables, in a separate memory-space. From the moment of its creation, the new process will also obtain its own running time from the process scheduler of the operating system, independent of the process from which it was created. This approach is for example be used by the *Terminal application* when executing a command, where the code-segment of the created process is replaced with the code-segment of the called executable [4].

The second approach is using threads, which is parallelisation inside a single process. The memory-space of the process is shared between the running threads. All variables existing before creating a new thread are accessible from within the threads. This makes the initialisation process less computationally expensive, since nothing has to be copied. Note however that this also requires correct synchronisation, since concurrent access of memory can easily lead to memory-corruption and race-conditions. Also, invalid behaviour of a single thread will end the whole process, and thus also all other running threads of this process. For the experiments in this dissertation, we only worked with multi-threading.

There is a distinction between **task-parallelism**, where each parallel instance performs its own task, and **data-parallelism**, where all parallel instances perform the same task but on different parts of the data. In section 4.4, section 4.5 and section 4.7 we perform data-parallelism, while in section 4.6 we focus on task-parallelism.

There are a lot of libraries available to work with threads. For our experiments, we used the OpenMP library [48], which uses pragmas in the code to define which shares of the code to run in parallel. The parallelisation we describe in section 4.6 uses the thread capability included into the Standard Library since the C++-11 standard [2]. Before this novel standard, the C++ language had no support for threads, and required the use of external libraries.

To determine the benefit of multiple threads, we up-scale the images to guarantee that sufficient layers are available to fill up the maximum of 32 threads. Note however that the platform we use has only 16 independent cores with hyper-threading, which evidently will not give the same performance as having 32 real cores.

4.3 Related work

Although parallelisation is commonly used in literature to make the pedestrian detection process faster, it is not always clearly described how it is applied. We can make a distinction between multi-core CPU parallelisation, where the number of threads is relatively small and thus more workload per parallel instance (known as coarse-grained parallelism), and GPU based parallelisation, which mostly benefits from fine-grained parallelism such that a very large amount of small tasks are evaluated in parallel. This distinction is purely based on the difference in device architecture between these two, which we describe in more detail in section 4.7.

In section 2.3 we described the FFLD detector [35]. To improve the detection speed of their approach, they evaluate the layers of the feature pyramid in parallel using OpenMP. We use a similar approach in section 4.4 as our first naive approach. In the latest release of the DPM code (DPMv5) [47] parallelisation is performed in two parts of the code. The first is evaluating all possible part placements in parallel. The second is convolving the different components, each representing a separate view of the object, in parallel with the whole feature pyramid, to obtain the initial root-scores. The work of Cho et al. [18] combines calculating the layers of the feature pyramid in parallel and evaluating the different components of the model in parallel.

When multiple models should be evaluated at each location, these can also be evaluated in parallel, which is performed by Sadeghi and Forsyth [81]. Each thread is assigned an equal number of object models to evaluate. To guarantee a high processing speed, the evaluation of the running threads is stopped after a certain time. The faster the threads are stopped, the higher the processing speed, but at the cost of accuracy since not all locations are checked for all possible object occurrences.

GPUs allow performing many calculations in parallel, and since most operations of pedestrian detection have to be performed on each location of the image, this is a beneficial task to implement on GPU. Prisacariu and Reid proposed the implementation of the HOG algorithm [19] on GPU [78], which was coined *fastHOG*. Their implementation is a factor of 67 times faster as the (non optimised single threaded) CPU implementation. A similar implementation can be found in the OpenCV library [14], and was also implemented by Sudowe and Leibe [84] who combined it with a ground constraint approach which was coined *groundHOG*. Benenson et al. [6] used a GPU implementation of the ICF detector [32] which is roughly 15 times faster as their CPU implementation of the algorithm. In their work, they use multiple models, each for a single scale of the object, instead of multiple layers. Each of these models has to be evaluated

on the same feature layer at each location. Since all models can independently by evaluated at each location, it allows great parallelisation opportunities which benefit from been executed on GPU. Their ICF implementation, with the use of multiple models and a soft cascade approach, reaches already a speed of 50Hz. To obtain an even higher processing speed, they use on top of this a ground plane constraint, which we explain in chapter 5.

4.4 Parallelisation of the feature pyramid

In this section we focus on the reduction of the *latency* between receiving the frame and the moment the detections are available. Reducing the latency is important in time-critical applications. For example, when the time to respond is very small, such as in traffic safety applications, we benefit from having the detections of a particular frame as fast as possible. Also, when the detection results of frames are dependent on each other, as is the case when using a tracking-by-detection framework (section 5.4), we can only benefit from parallelisation by making the processing of single frames faster. In section 4.5, we improve the *throughput* by evaluating multiple frames in parallel instead. Although the amount of frames that can be processed per time unit is improved, the time to process a single frame (the latency) will not change.

For our experiments in this section we use the OpenMP library [48], which makes running the iterations of a loop in parallel quite easy. The iterations are randomly assigned to the available threads. As a first experiment, we evaluate the layers of the scale-space pyramid in parallel. Since all these layers are independent of each other, no communication is required. The detection results of each layer are written to an independent memory space for each thread. When all layers are processed, the results are merged sequentially and are subjected to *Non-Maximum-Suppression*. The reduction of the processing time in relation with the number of used threads is shown in figure 4.2.

We can observe that the reduction in calculation time does not scale linearly with the number of threads. The main cause for this is that the workload is not equally divided over the threads, since the layers of the scale-space pyramid are not equal in size (hence the "pyramid"). Since we know up front the rescaling between these layers and the number of threads to use, we can cluster the layers to obtain a more evenly distributed computation load over the threads. Figure 4.3 visualises this assignment when 3 threads are used.

As we can see, the workload is not yet completely even distributed over the threads since each layer has a quantised size. When using this approach, we obtain the results shown in figure 4.4. Compared with the previous approach,

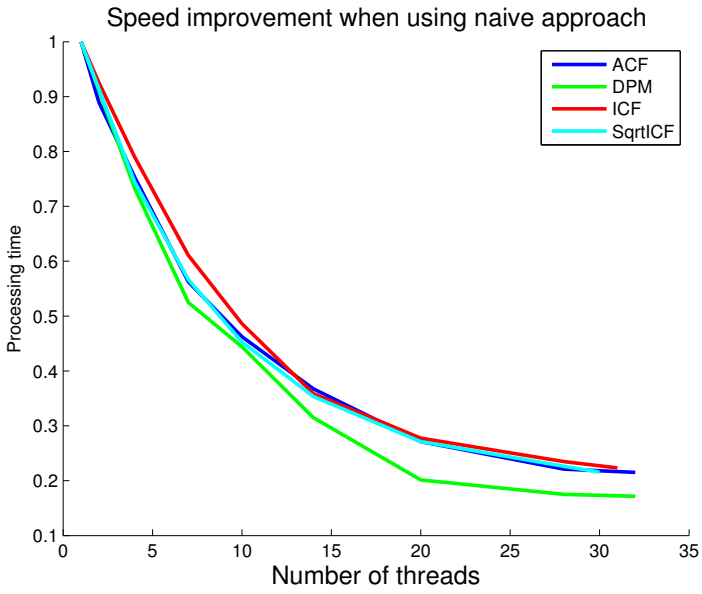


Figure 4.2: Speed improvement when using a naive approach to evaluate layers of the scale-space pyramid in parallel.

visualised with dashed lines, we obtain a far more efficient use of our threads especially under 10 threads. The more threads we use, the closer we fall back to the previous case where the "best" assignment is that each thread processes a single layer. Note that the approach we use here has only the goal to obtain a more evenly distributed workload over the thread. In [44], the authors describe a slightly more complex scheduling problem, which turns out to be NP-complete in most cases. We however are currently only interested in a solution that is easy to calculate with a better balance of computational resources than the naive approach we described before.

4.5 Parallelisation of frames

In this section we will discuss how to optimise the throughput of the pedestrian detection algorithm. The throughput represents the number of frames we can process per time unit. The approach we used in section 4.4 will both decrease the latency and the throughput. When the latency is reduced, and thus the processing of a single frame takes less time, the number of frames we can process

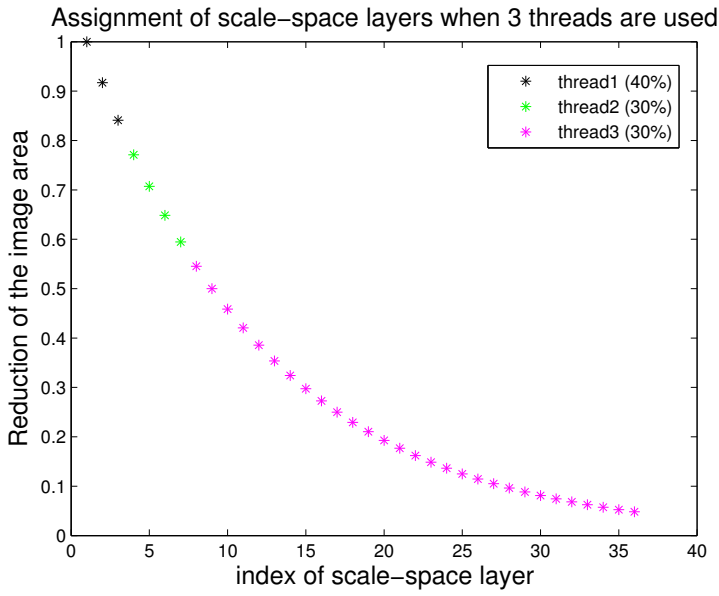


Figure 4.3: Improved assignment of the layers when using three threads.

per time unit will evidently also increase. An improved throughput does not guarantee an decreased latency though.

Instead of evaluating a single frame at a time, which we described in chapter 2, here we will evaluate multiple frames in parallel. Figure 4.5 shows the reduction in processing time per frame in comparison with our optimised parallelisation approach we described in section 4.4 (in dashed lines).

We can see that evaluating multiple frames in parallel is slightly more beneficial compared to evaluating the feature layers in parallel while using balancing of the workload. An important remark however, is that sufficient data needs to be available for both cases to work efficiently. As we mentioned earlier, we had to do a large initial up-scale of the frames to generate sufficient scale-space layers to process in parallel. In the application of evaluating all frames in a directory offline, it is easier to keep all threads busy when processing the images in parallel instead. Evidently both parallelisation techniques can be combined.

For example, in the application of blurring the faces of pedestrians (section 1.2.1), both techniques can be applied, since the high resolution implies many layers in the scale-space pyramid, and there is a large number of images. Note

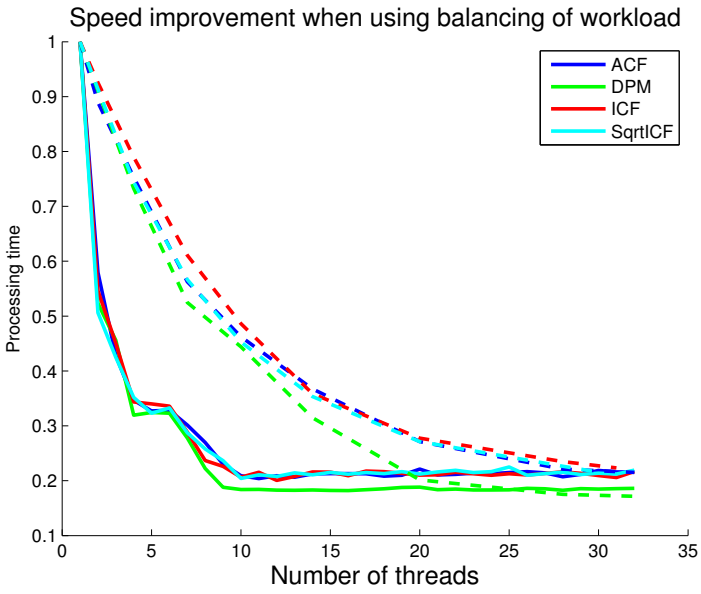


Figure 4.4: When we divide the workload more evenly over the threads, the parallelisation becomes more efficient (solid lines) compared to the naive approach with random assignment (dashed lines).

however that this application allows heavy search space reduction (chapter 5), which will decrease the amount of layers in the scale-space pyramid. As another example, the application of the detection of pedestrians around AGVs (section 1.2.3) requires small latency to increase the time to respond. Therefore evaluating the layers of the scale-space pyramid in parallel will be the best option.

4.6 The use of an assembly line architecture

In our publications "Pedestrian detection at Warp-speed: exceeding 500 detections per second" [25] and "On-board real-time tracking of pedestrians on a UAV" [22] we made use of task-parallelism. In these publications, we divide the task of the application into subtasks (e.g. read frame, calculate feature pyramid, search for model occurrences in the feature pyramid, sort the frames, ...). Each subtask is assigned to a separate thread that will execute the same task as long as the application runs. Processing a single frame can be seen as passing it

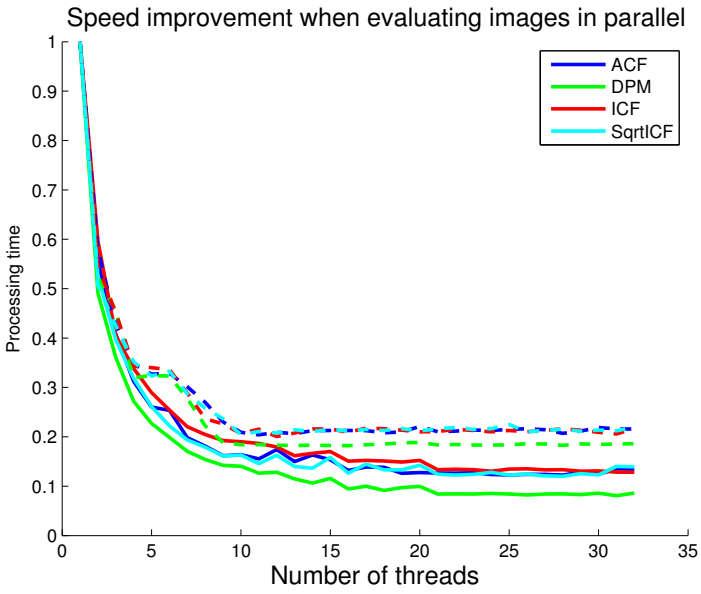


Figure 4.5: Comparison of evaluating multiple frames in parallel (solid lines) and our optimised technique for evaluating layers of the feature pyramid in parallel (dashed lines).

through an assembly line, where the data of the frame is passed from subtask to subtask with the help of queues. In chapter 6 and chapter 7 these publications are more deeply described, including the division of subtasks for those specific applications.

Note that the threads are now dependent on the previous stage for their input. This implies that the slowest subtask determines the throughput (*"the chain is only as strong as the weakest link"*).

4.7 Using GPU hardware

In section 4.4 and 4.5 we discussed how multi-core CPUs help to improve the processing speed of our pedestrian detection algorithms. In this section, we exploit the data parallelisation opportunities that Graphical Processing Units (GPUs) can offer. In contrast to CPUs, where the amount of cores is rather limited, GPUs have a very large amount of cores, which makes them perfect for

Figure 4.6: Difference in design philosophy between CPUs and GPUs, image taken from [58].

massive fine-grained parallelism. Note however that this large amount of cores on a GPU comes at the cost of making them less advanced, and so they lack features such as branch prediction, large caches,... This difference is due to the difference in design philosophy between the two, which is visualised in figure 4.6. Where the CPU is designed for running complex applications mostly sequential, the GPU is designed to perform relatively simple instructions on large amounts of data. To be able to transfer this amount of data, the memory bandwidth that can be reached on a GPU is approximately six times the memory bandwidth of available CPU chips [58].

The content of this section is based on the material we published in "Is the game worth the candle? Evaluation of OpenCL for object detection algorithm optimization" (PECCS 2012) [23], which was later extended to "Faster and more intelligent object detection by combining OpenCL and KR" [24] as a journal publication. We will describe our GPU optimisation of the feature pyramid of *Deformable Part Model* detector [41] to optimise the processing time, and thus the latency, of this algorithm. We selected this algorithm since it is the slowest when evaluated on CPU, and requires the largest speed-up. Note however that the use of GPU hardware is equally beneficial for other pedestrian detection algorithms.

We used the OpenCL programming language, at the time of publishing a novel standard for heterogeneous programming of devices, including GPU hardware. For the case study we describe in chapter 6, we ported this OpenCL implementation to CUDA, since this turns out to be slightly more efficient when executed on Nvidia hardware.

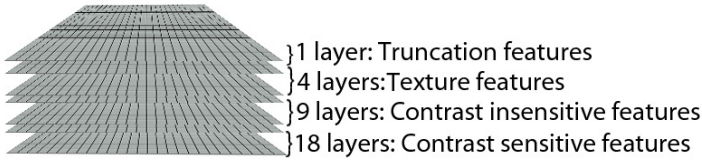


Figure 4.7: The feature types used by DPM, which are calculated for each layer of the scale-space pyramid.

The algorithm can be divided in two main parts:

1. The construction of the feature pyramid.
2. Model evaluation, the search for a pre-trained model in the feature pyramid.

We will focus on the implementation of the first part. The calculated feature pyramid is independent of the model we are looking for. Our optimized implementation can thus be used in a detector for any arbitrary object class, as long as a pre-trained model is available: pedestrians, bicycles, horses, cars, ...

The construction of the feature pyramid can be subdivided in three stages:

1. Rescale the image to different resolutions of the same image, resulting in a scale-space pyramid. This allows to find the model on different sizes without the need to rescale the model.
2. Create histograms of the orientations of the gradients of the pixels (HOG). Using the gradients creates invariance for illumination changes.
3. Calculate additional features based on these HOG features.

Figure 4.7 shows the function of the 32 feature types that are calculated for each layer in the scale-space pyramid. Each kind of feature emphasizes a specific property that can be used to distinguish possible detections from negatives

Once the feature pyramid is built, it can be used to search for a model on different scales. Each model exists of a root model, which is used to find the object as a whole (comparable to the model used by Dalal and Triggs [19]), and multiple part models. The part models, used to detect small parts of the object whose position can vary with respect to the root model, are searched for at twice the resolution of the root model. The higher resolution offers more image information since more pixels of the same image area are present.

4.7.1 Implementation

In this section we go deeper into our OpenCL specific implementation details of the feature pyramid. We will explain how the different parts work, and point out the advantages and disadvantages of our implementation. Our OpenCL implementation is based on a C-port of [40]. We later refactored this implementation to C++ to obtain the implementation we described in chapter 2. The OpenCL kernels are written in a language based on the C99 standard, so using a C-implementation is a good starting point.

OpenCL

Modern computation platforms typically include one or more CPUs, GPUs, DSPs, ... All these hardware types are designed and optimized for a specific type of calculations. Optimized hardware leads to faster execution and/or less power consumption, so it is beneficial to use the most optimal hardware as much as possible. The problem is that each kind of hardware has its own instruction set, and so requires very specific programming. OpenCL, Open Computing Language [56], is a novel open standard for heterogeneous computing. It is a framework for writing programs that can use these platforms in a heterogeneous way, in contrast to CUDA, which was developed by Nvidia, specifically for its own GPU hardware. This allows to write an efficient and portable implementation of an algorithm which exploits the possibilities of each part of the algorithm on the most suitable device (multi-core CPU, GPU, cell-type architectures or other parallel processors). Since it is heterogeneous, we do not have to know in advance which hardware will be used to execute the algorithm. The used platform can easily be changed using an initialisation variable of the program. Since different devices have different instruction sets, the compilation of the OpenCL kernels can even happen at runtime. Here we focus on the exploitation of data parallelism opportunities of an object detection algorithm. Since GPU hardware is optimized for data parallelism, this is the hardware we will use.

The code is written in the form of kernels. A kernel is a block of code, written in a language based on C99, that can be executed in parallel. For example, when each element of a matrix has to be multiplied by a certain value, the kernel may contain the code for one multiplication and this kernel will be executed for all elements of the matrix. The execution of the NDRange (all threads that execute the kernel code) is divided in workgroups. A workgroup is subdivided in work-items, which will execute the kernel code in parallel (figure 4.8). To distinguish different execution threads, each thread has a unique global id, and within a workgroup each thread has a unique local id. Both are assigned for each dimension.

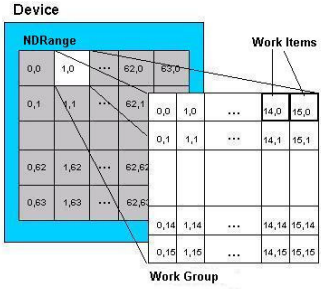


Figure 4.8: The execution of the kernels is divided in workgroups, which can be subdivided in work items. Each work item executes an instance of the kernel.

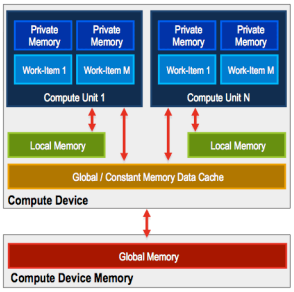


Figure 4.9: Memory model of a GPU, from slow to fast: Global memory, Local memory, Private memory.

Figure 4.9 shows the memory model of a GPU device. The memory access times are going from the slowest at the bottom (starting with the memory of the host computer) to the fastest at the top (private memory). While developing, it is beneficial to use the fastest memory as much as possible. The global memory of the GPU (and CPU) is shared over all executing work items, the local memory is shared over the work items in the same workgroup and the private memory is only accessible by the running work item.

Rescale

The first step in the construction of the feature pyramid is the construction of a scale-space pyramid, which contains rescaled versions of the input image. This allows to detect the model at different sizes, since for each layer the features will be computed, which will later be used for model evaluation.

We used two implementation approaches: a straightforward implementation of linear interpolation, and an implementation using dedicated GPU functions for rescaling. In both implementations, we launch one thread for each pixel of the destination image. This allows for maximum parallelization.

Since the rescaling can be divided in vertical and horizontal rescaling, the kernel executed by each thread is split up in these two directions. At the beginning of the thread, we calculate which pixels will be needed by the linear interpolation process. These pixels are then used to rescale vertically and the result is stored in private memory (registers of the GPU, memory with the best access time). In

the next step, these vertically rescaled pixels are used in a horizontal rescaling step which results in the destination pixel. The calculated pixels are written to global memory as part of a layer in the scale-space pyramid. The disadvantage is the non-linear access pattern of the needed pixels. This problem can be solved by using texture memory.

Texture memory is a special kind of memory that can be used by GPU hardware. It is optimised for a more random access pattern and it is cached. Since GPUs are mostly used for processing image content, certain functions are very frequently used, such as image rescaling. To speed up these functions, a hardware-optimised version is present on most GPUs. The use of texture memory allows the use of these functions. Since not all OpenCL-capable hardware supports the use of texture memory, it is not included in the OpenCL specifications, which makes the code only usable on a GPU platform.

Histogram

When the scale-space pyramid is built, we can calculate the HOG features for each layer of the pyramid. For this task, the image is divided into blocks of 4 by 4 or 8 by 8 pixels, and for each of these blocks a histogram is calculated based on the orientation of the gradients of the pixels in the block. A pixel gradient is based on the horizontal and vertical derivative, which are obtained by subtracting two subsequent horizontal or vertical pixels. In the case of colour images, only the strongest (largest) gradient of the colour channels is used for the histogram. To determine the orientation of the gradient, the horizontal and vertical derivatives are multiplied with respectively the cosine and the sine of the bin orientation (the use of 18 bins results in 20 degrees per orientation bin) and are then summed. The maximum response gives us the orientation of the gradient.

To improve the robustness, a linear interpolation scheme is used, such that each pixels contributes to four histograms. Hereby the size of the contribution depends on the distance between the pixel coordinate and the histogram centre. The construction of the well known SIFT local feature descriptor [63], which is also based on HOG, uses a very similar approach. This interpolation scheme makes it difficult to divide the histogram into smaller histograms. This restriction makes this part of the algorithm the most difficult to parallelise and even the most computation intensive task of the feature pyramid.

Each histogram contains the votes of a limited amount of pixels (4 by 4 or 8 by 8). When we would use a similar approach like in the rescaling part, and use one thread per voting pixel, we face the problem that multiple pixels need to have write access to the same memory addresses. We found out that the classic

solution of using a semaphore to lock a memory location is very complex to implement on GPU, since the program counter of multiple threads is shared for performance reasons. Sharing of the program counter has the effect that the code for waiting on the release of the lock is shared with the thread that has the lock, so the lock is never released which results in a deadlock. An extra disadvantage of the semaphore approach is that it is against the philosophy of parallel programming because we create a bottleneck by waiting for the release of the memory lock, which prevents gaining computation speed by parallel execution. Our solution to this problem is to keep the four histograms a group of pixels will vote for, separate in memory, and launch one thread for each block of pixels which are voting in the same histogram. Each thread will process all the pixels that vote in the same 4 histograms sequentially. With this approach the kernels do not have to wait to write their result. When the four groups of histograms are filled in, they can be summed together, with respect to their misalignment, to get the final histograms (HOG features) of the image layer.

Calculate additional features

The feature values where the model will be evaluated on, are calculated by the "additional feature" kernels, which includes the required normalisation of the HOG-features to obtain the 18 layers of *contrast sensitive features*. As we noted earlier in figure 4.7, there are 4 types of features.

For the calculation of the additional features, we chose the number of threads launched to be equal to the number of places in a feature layer. So each thread calculates 32 values.

With the techniques described above, we now have OpenCL implementations of the scale-space pyramid (Rescale), histogram and calculations of the additional features, which are ready to be tested and compared.

4.7.2 Experimental timing results

In this section we will present the timing results from different experiments. We will begin with our reference implementation on CPU and go step by step to a total implementation of the feature pyramid in OpenCL.

Figure 4.10 visualises the step-by-step conversion from a pure C-implementation to an OpenCL implementation of the feature pyramid. Each time an extra part of the algorithm is performed using OpenCL on GPU, while the remaining part is still executed on CPU.

Function	Share of calculation time (%)
Rescale	20.36
Histogram	69.39
Additional features	10.25

Table 4.1: Distribution of calculation time of the feature pyramid calculation on CPU.

Experiment specifications

All experiments are executed on the same platform, with a core i7 965 (3.2 GHz) CPU and a dedicated Nvidia GeForce GTX 295 GPU. This GPU has the possibility to be used as two parallel devices, but we only use one. We run our experiments under the linux operating system. Note that the speed results we provide here are *not* obtained on the same hardware as we use for other experiments, except for the case study we describe in chapter 6, which builds further on this implementation.

The experimental timing results we used the PETS2010 dataset [77] using images with a 600x480 resolution. For OpenCL profiling we used the visual profiler released by Nvidia.

C implementation

The CPU implementation of the algorithm is used as a reference. Since OpenCL is an extension of the C programming language, using a C-port as a starting point is of great use. In table 4.1 the division of the calculation time for the CPU-implementation can be observed. The largest share of the computations is consumed by the calculation of the histogram.

Rescale in OpenCL

As a first experiment, the rescaling of the images is executed on the GPU. The source image is transferred one time to the GPU, and used multiple times to construct the scale-space pyramid. The resulting scale-space pyramid needs to be transferred back to host memory for further processing. In figure 4.10 one can observe that the amount of time spent transferring information is relatively large compared to the actual computation time on GPU, namely the rescaling of the images. However, figure 4.10 shows a large speed improvement (second bar from the left).

Rescale and Histogram calculations in OpenCL

In this second experiment we execute two parts of the feature pyramid on GPU, namely the rescaling of the images and the calculation of the Histograms (HOGs) from these images. After calculating the histograms based on the gradients of the images from the scale-space pyramid, these are transferred back to host memory for the calculation of the additional features, which is performed on CPU.

In figure 4.10 we can observe that almost all computation time of the CPU implementation is consumed by the rescaling and the calculation of the histogram (table 4.1). According to Amdahl's law, replacing them with a faster alternative, which is what we do, will result in the largest time profit.

Total feature pyramid in OpenCL

In our final experiment we execute the total feature pyramid on GPU. The initial image is transferred to device memory and after the execution of all the kernels the total feature pyramid is transferred back to host memory.

Total feature pyramid in OpenCL using texture memory

As mentioned before, using the GPU as execution platform allows the use of dedicated functions by using texture memory. In this implementation we use this texture memory to speed-up the rescaling of the images.

As we can see in figure 4.10, the use of dedicated functions decreases the execution time of the rescale kernels to a minimum. Note however that using these dedicated functions of the GPU make this OpenCL implementation no longer portable to other OpenCL capable devices.

Comparison of results

In figure 4.10, a comparison of the experimental timing results is given. We can observe that the use of dedicated hardware results in a feature pyramid over six times as fast as the CPU implementation. We can also notice that the largest speed up is obtained in the parts that are most computationally intensive on CPU, namely the image rescaling and the calculation of the histograms. The speed we gain by implementing functions in OpenCL is almost directly proportional to the time needed on CPU. We obtained a frame rate

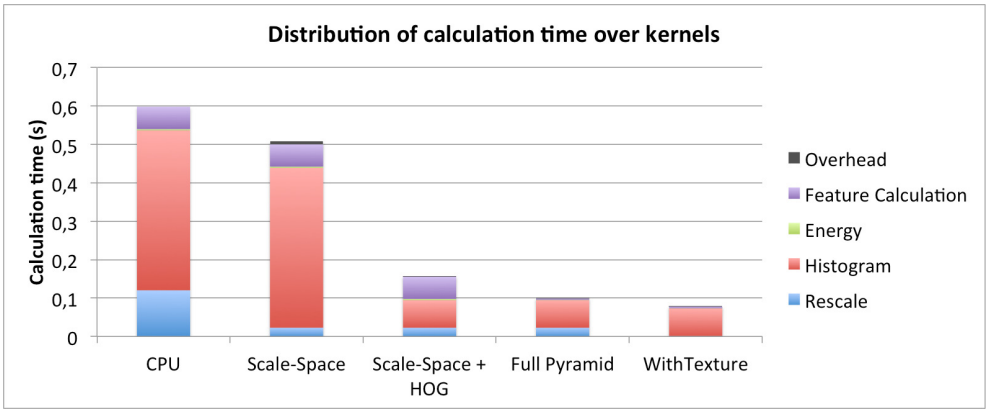


Figure 4.10: Distribution of calculation time on GPU over kernels

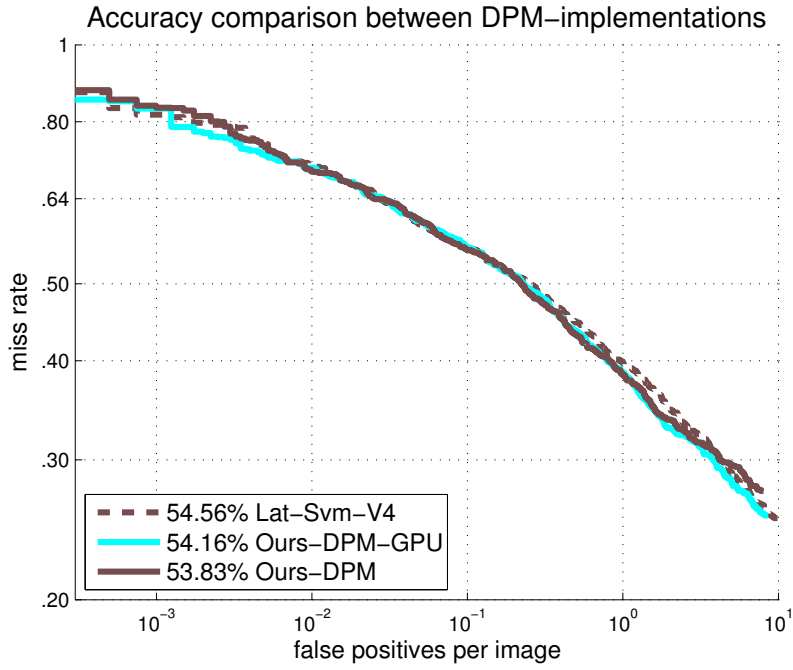


Figure 4.11: Accuracy comparison between the original Matlab-implementation [40], our CPU implementation and the implementation using GPU hardware for the feature pyramid of the *Deformable Part Model* detector.

of approximately 2 fps for a complete pedestrian detection (on all scales) on the PETS dataset. In chapter 6 we will exploit parallelisation further with the use of multiple threads on CPU, in combination with a generalised ground plane estimation approach and a tracking-by-detection approach that we both describe in chapter 5.

Although we obtain a higher detection speed, it is important that it does not come at the cost of detection accuracy. In figure 4.11 we show the accuracy comparison on the Caltech dataset [33] between our CPU version, which we used before, and when the feature pyramid is calculated on GPU. As we can see, our GPU port does not come with an accuracy loss.

4.8 Conclusion

In this chapter we discussed how the use of parallelisation can benefit the processing speed of pedestrian detection algorithms. In section 4.4 we decreased the latency between receiving a frame and delivering the detections of that frame, by evaluating the layers of the scale-space pyramid in parallel. We found out however that a naive approach did not scale well, since the layers of the scale-space pyramid are not equal in size, and so the workload is not evenly balanced between the threads. We improved this implementation by dividing the workload ourselves, since we can determine the sizes of the scale-space pyramid upfront. This drastically improved the scalability.

In section 4.5 we mainly focussed on the improvement of the throughput. Here multiple frames are evaluated in parallel, and so the processing time of a single frame remained the same. This led to a slightly more efficient use of the available threads compared to the optimised approach of evaluating the layers of the scale-space pyramid in parallel, but may be less applicable for most applications.

In section 4.6 we described the use of task-parallelism, where the application is divided into subtasks which each are assigned to a separate thread. We apply this approach for the case studies we work out in chapter 6 and chapter 7.

In section 4.7 we demonstrated the benefit of using GPU hardware, by improving the frame-by-frame detection speed of the *Deformable Part Model* detector by implementing the feature pyramid on GPU. Our GPU implementation obtained a speed-up of a factor 6 over the CPU implementation of the feature-pyramid calculation, without losing accuracy.

Although we did improve the calculation speed of the feature pyramid with six times, we do not yet fully exploit the capabilities of the GPU. Our approach of iteratively replacing parts of the calculations with a faster OpenCL equivalent

comes at the cost that we make extensive use of global memory to store intermediary results. Since the speed of global memory is roughly 100 times slower than for local memory, an implementation that keeps intermediary results in faster memory, avoiding global memory access, will reach higher speed-ups.

Chapter 5

Search Space Reduction

5.1 Introduction

In chapter 2 we described the classic approach for pedestrian detection as a search at each location and at multiple scales of an image, which has a high computational cost. In the previous chapter, we focussed on hardware based optimisations by exploiting parallelisation opportunities. In this chapter we take another approach to improve the detection speed: we reduce the search space by integrating knowledge about the scene. A reduction of the search space will evidently lead to faster processing.

In section 5.2, we will give an overview of the related work on this topic. In section 5.3 we will reduce the number of scales to search for pedestrians based on the heights we can expect, and thereby reduce the number of layers in the feature pyramid. In section 5.4 and section 5.5 we take this approach a step further by respectively integrating temporal information in the form of a tracking-by-detection framework, and the integration of the ground constraint.

5.2 Related work

In chapter 2 we already gave an overview of the literature on pedestrian detection algorithms. The sliding window approach these detectors perform, requires an exhaustive search for model occurrences on all possible locations (and multiple scales) in the image. The combination with an increased pedestrian model complexity, such as the approach of [42] who extends the HOG-model [19] with

deformable parts, makes this infeasible for fast processing. However, extensive research has been dedicated to the topic of improving the detection speed by reducing the search space, possibly in combination with exploiting parallelisation opportunities.

In the work of Benenson et al. [6] multiple optimisation approaches are used which we already described in the previous chapter: the use of multiple models instead of evaluating the image at multiple scales and an optimised GPU implementation of the ICF detector. On top of that they exploit scene constraints by using a *Stixel-world* approach [9], calculated from a stereo-setup. This allowed to obtain detections at over 100 frames per second. Later this *Stixel-world* approach was improved, allowing pedestrian detection at 165 frames per second [5]. Sudowe and Leibe [84] show how a sliding window based detector can efficiently benefit from a ground plane constraint. They demonstrate this with a CUDA implementation of the HOG-algorithm (groundHOG), similar to [78]. Basically, based on a homography of the ground plane, for each possible real-world size range of a pedestrian (e.g. 165cm to 185cm), the area in the image is determined where the corresponding detections can be found. Also Cho et al. [18] demonstrate a similar approach, to achieve real-time pedestrian detection results on the Caltech dataset.

A lot of tracking approaches are known in literature, and each year new improvements are found. Pflugfelder et al. [59] presented a benchmark to compare different tracking approaches, aiming at single-object model-free tracking approaches. This allows making a fair comparison between tracking approaches for the same dataset(s). In this dissertation, we focus on tracking-by-detection, where the probability of the tracker is based on the use of a detection-based approach. In [49], the authors use online boosting to create a model from the detections over time to cope with possible variations of the object. In [70] a combination is made of a Mixture Particle Filter [88] based on colour information, and an offline trained detector. In particle filtering, the current position of an object is determined based on a finite set of Dirac measures, where each measure is weighted based on the probability of the object being at that location. The higher the number of samples, the more accurate the position of the object can be determined. The computational requirement of the particle filter depends on the number of samples (particles) used, and the computational requirement of the technique to determine the weight for each particle. The mean-shift tracker, which was originally presented in [43], is a technique to determine the maximum of the Probability Density Function (PDF) of the object's location. It will iteratively narrow down to the maximum, where in each iteration the mean of a kernel window is determined, until the mean converges. The efficiency of the mean-shift algorithms, just as with the particle filter, depends on the technique to determine the samples for the PDF.

In this chapter, we use the Kalman filter as a tracker [55], which we use in combination with a detection algorithm. In this algorithm only a single hypothesis of the object's location is used. We give a more detailed overview on how the Kalman filter works in section 5.4. In "Towards robust automatic detection of vulnerable road users: monocular pedestrian tracking from a moving vehicle" [86], we make a comparison between the use of the Kalman filter and the Particle filter, where the probability of the samples is determined by evaluating the *Deformable Part Model* detector for each particle. Although the particle filter turns out to be more accurate, the overhead of running the DPM detector for each particle comes at a large computational cost. In chapter 7 we will use a particle filter based on colour histograms, which are less computationally intensive.

In [45, 46, 97] a tracking-by-detection framework is used similar to the one we describe in section 5.4, to analyse traffic behaviour. The per-frame detections of the *DPM* detector [41] are tracked using a Kalman tracker, where the detection association is made based on the *Hungarian method* [61]. We will extend this approach to reduce the search space, similar to the approach we describe in "Towards robust automatic detection of vulnerable road users: monocular pedestrian tracking from a moving vehicle" [86], although in this paper the association was not yet made using the Hungarian method.

5.3 Reduction of the Scale-Space Pyramid

A first step that can be taken to reduce the search space, is limiting the amount of scales the model is searched on. The size a pedestrian will appear at in the image is dependent on the distance from the lens, and the observed scene. We will explain this technique based on the annotation data of the Caltech pedestrian dataset [33], so note that the settings for other datasets or applications will be different.

Figure 5.1 shows the distribution of the annotations in the Caltech dataset under the *Reasonable* evaluation criterion (pedestrians are at least 50px height and 65% visible) at the bottom. We can see that the largest annotation is below 355px. So instead of evaluating all the scales between 50px and 480px, we only need to search for pedestrians between 50px and 355px. This figure also shows the distribution between TPs and FPs for the different detectors on the 0.1 FPPI point. We can point out that limiting the number of scales will improve the accuracy of the *HOG* detector, since all false detections above 355px will be avoided. The accuracy improvement is negligible though. Table 5.1 shows the speed improvement we obtain using this reduced search space.

Technique	Model size	Full range	Reduced range (50px-355px)	Speed-up
Ours-HOG	96px	0.58 fps	0.68 fps	1.17x
Ours-ICF	100px	0.68 fps	0.70 fps	1.03x
Ours-DPM	120px	0.32 fps	0.34 fps	1.06x
Ours-ACF	100px	6.90 fps	7.22 fps	1.05x
Ours-SqrtICF	100px	6.78 fps	7.04 fps	1.04x

Table 5.1: Comparison of the detection speed when using scale-space reduction.

The speed improvement we obtain by reducing the number of scales is dependent on the scales that are pruned. In section 2.5 we gave the processing speed per detector for multiple resolutions, and here we saw already a big difference between VGA resolution and twice this resolution. Figure 5.2 visualises the number of windows that need to be evaluated for a specific scale (multiplied with the VGA resolution to obtain the resolution of that specific scale-space layer). The marker points indicate at which scales the model would be evaluated if 8 scales per octave are used, which is the case for *Ours-ACF* and *Ours-SqrtICF*. When more scales are used per interval, such as for *Ours-DPM* where 10 scales are used per octave, we obtain the same function course, but the marker points are denser. When pruning the scales for the larger pedestrian sizes, the marker points at the bottom left of the graph in figure 5.2. This is not so effective as pruning scales at the smaller pedestrian sizes (at the right of the curve), which clearly explains the large decrease in detection speed we experienced in table 2.1 when up-scaling the image.

The importance of scale range becomes clear when the connection is made between the pedestrians height and the time to respond. This relation is visualised in figure 5.3. In figure 5.3a is shown how the height of the pedestrian decreases when further away from the camera, based on the pinhole model. We will use this relation in section 5.5.2 to reduce the search space based on the ground constraint. In figure 5.3b, this relation is shown for the pedestrian's height in pixels and the distance from the camera, based on the Caltech dataset. Also, the time a car has to stop (until collision) is shown when the car is driving at a speed of 55 km/h (according to [34]). This figure shows that e.g. when we have a system that requires 4 seconds to respond (including all subsequent actions such as braking), pedestrians should be already detected when they are 30 px high. If 1.5 seconds is sufficient, we only need to detect them at 80 px, which allows reducing the number of layers to evaluate, and as such the computational requirement.

In this section we gave a first step to reduce the search space for a detector, by integrating knowledge on the size of pedestrians in the scene. This allows

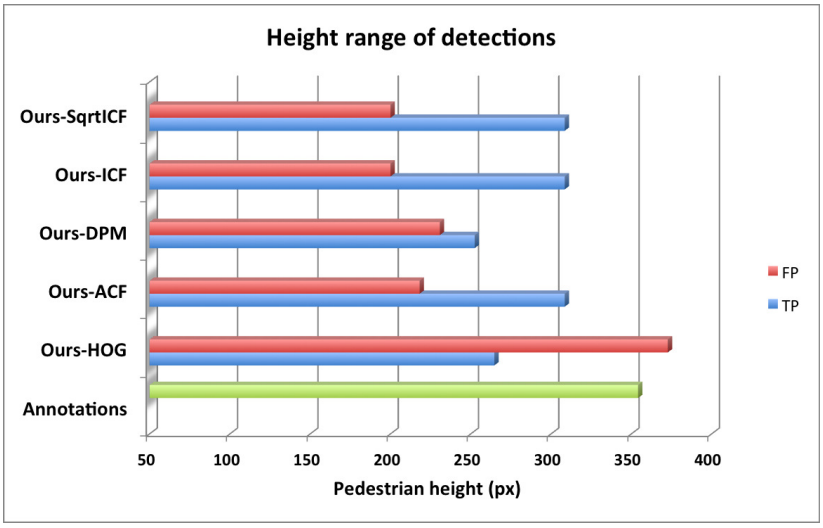


Figure 5.1: The height range of the detections.

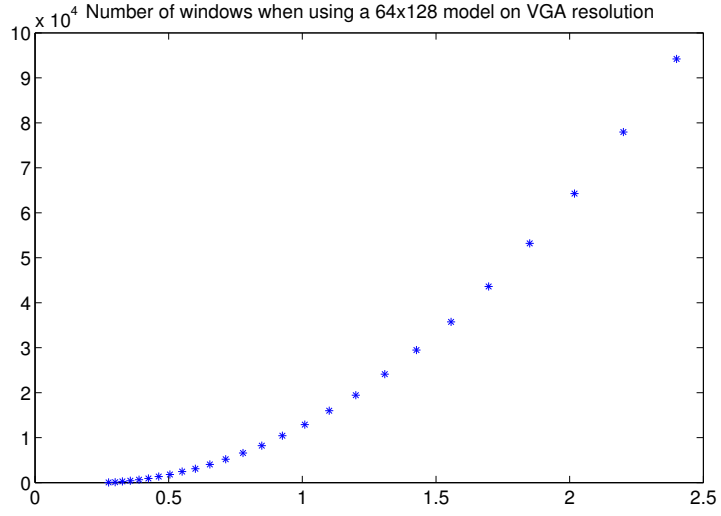


Figure 5.2: The number of windows that need to be evaluated for each scale-factor on an VGA image. To obtain the image resolution for a specific scale, multiply the VGA resolution (640x480) with the scale-factor.

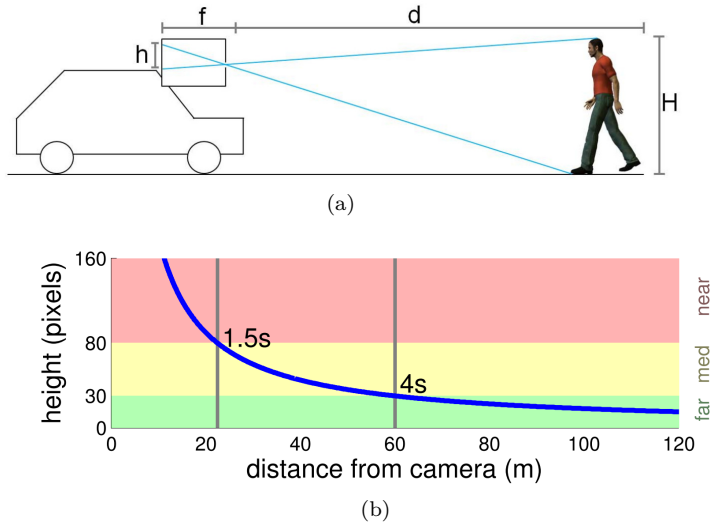


Figure 5.3: The relation between the distance from the camera and the height in pixels of the pedestrians in the Caltech dataset. Taken from [34].

reducing the number of sizes a pedestrian is searched for, and as a result the processing time. In section 5.4 and 5.5, we extend this approach by integrating temporal information and exploiting the ground constraint respectively.

5.4 Integrate temporal information

In the previous section, we presented a technique on how to reduce the number of scales, and thereby the search space for pedestrians. In this section, we will integrate temporal information in the form of a Kalman [55] based tracking-by-detection framework. A tracker is used to follow pedestrians over multiple frames, and thus extends the information from its 2D position in a single frame we used so far, to a sequence of positions in consecutive frames. The Kalman tracker, which we explain in more detail in section 5.4.1, follows a two step loop to keep an internal state as close as possible to the observed process (in this case the trajectory of a pedestrian). In the *Prediction step* the next location is predicted based on information from the previous frames and thus its current internal state. In the next step, the *Correction step*, the internal state is updated proportional to the difference between a (noisy) measurement of the current

process and the predicted location. In our case, the measurement is running a pedestrian detector to determine the current position of the pedestrian.

The tracking-by-detection approach we describe in this section was used, in a similar form, in our papers "Towards Robust Automatic Detection of Vulnerable Road Users: Monocular Pedestrian Tracking from a Moving Vehicle" (ATINER 2011) [86], where the Kalman approach was compared with a particle filter [15], and "Pedestrian detection at Warp-speed: Exceeding 500 detections per second" (Embedded Vision Workshop 2013) [25] where the capability of search space reduction is used in combination with a hybrid GPU/CPU implementation of *Deformable Part Models* to obtain 500 detections per second. Both of these approaches where made in the context of the detection of vulnerable road users in the blind spot area of a truck, although the same technique can be applied in another context as well (as we will demonstrate here).

In "Towards Robust Automatic Detection of Vulnerable Road Users: Monocular Pedestrian Tracking from a Moving Vehicle" [86], the contribution of Floris De Smedt was to show the potential of GPU hardware to speed-up this framework. The majority of this paper, including the comparison and implementation of both a Kalman based tracking-by-detection framework and a particle filter in Matlab, where contributed by K. Van Beeck as first author.

Combining a hybrid GPU/CPU implementation of the *DPM* pedestrian detection, with a tracking-by-detection framework in "Pedestrian detection at Warp-speed: Exceeding 500 detections per second" [25] was contributed by Floris De Smedt, although it was not yet applied in an actual application. The work of K. Van Beeck on the detection of vulnerable road users in the blind spot area of trucks, formed a good application for this approach. The required technique of the *Warping Window* approach, and applying it on the Caltech dataset [34] was contributed by K. Van Beeck. Note however that the (C++) implementations of the tracking-by-detection framework in this chapter are developed from scratch by Floris De Smedt.

5.4.1 The Kalman tracker

We use a tracking-by-detection framework based on the Kalman tracker [55]. Here we will give a short insight in how this tracker works. The Kalman filter keeps an internal state, which is represented as a vector of variables. In [86], we used the following state:

$$x_k = \begin{bmatrix} x & y & v_x & v_y \end{bmatrix}^T \tag{5.1}$$

where the x and y represent the coordinate of the centre of the pedestrian, and v_x and v_y represent the velocity in both directions through the scene. Note that we are modelling the positions and velocities in the 2D image plane and not in the real world. To get this internal state as close as possible to the observed process, in this case the trajectory of the pedestrian through the scene, a two step loop is used which is visualised in figure 5.4. In the *Prediction step*, the current state is used to make an estimate of the next state using the transition matrix.

The transition matrix accompanying the state we noted earlier (5.1) is

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

which indicates that v_x and v_y are modelled as a constant value, while the coordinates x and y will change dependent on the value of v_x and v_y respectively. So in this case we are working with a constant velocity model. After the prediction, a measurement is made of the current process (the current position of the pedestrian). In the *Correction step*, the difference between the made prediction and the current measurement is used as an error value to calculate the Kalman Gain, which will on his turn be used to update the state.

We use the R matrix, which represents the *measurement noise covariance*, to determine how strongly the measurement is "believed", and thus how strongly the difference between the prediction and the measurement is taken into account [90].

The use of a tracker allows to overcome intervals where the pedestrian is not detected, which can be caused by a too severe deformation of the pedestrian such that the similarity between the features and the model is not sufficient to be categorized as a detection, or that the person is occluded due to other pedestrians or objects. Based on the state of the tracker, a thoughtful guess can still be made about the pedestrian's location.

5.4.2 Kalman Tracking-by-detection framework

To combine pedestrian detection and pedestrian tracking into the same framework, we evaluate the detector on each frame, which results in a list of detections. In [45, 46, 97] a full frame detection is used, while the technique we describe here will reduce the area the detector has to be evaluated.

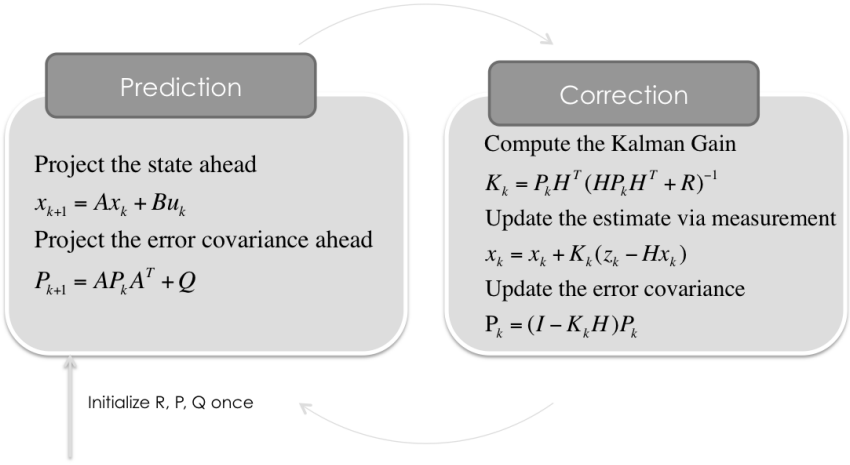


Figure 5.4: The two-step cycle of the Kalman-filter.

For each detection there are 2 possibilities:

- there is currently no tracker running for this pedestrian, in which case we will initialise a new tracker.
- there is already a tracker for this pedestrian, in which case we use the detection as a measurement for the correction step of this tracker. Here we also reset the lifetime of the tracker. The function of the lifetime will be explained later.

So, after the first frame, there is a large chance we have a list of running trackers, each belonging to a pedestrian in the scene. Also for each tracker there are two possibilities:

- There is a match between the detection and the tracker, which is the same as the second case for the detections. The tracker will use the detection as a measurement for its correction step.
- There is no detection matching the tracker, in which case the predicted position of the tracker is used as a "measurement" to correct the tracker's internal state. Also the lifetime of the tracker is decreased. When the lifetime of a tracker becomes zero, the tracker is removed. The lifetime solves the problem of *dangling trackers*, which are trackers that have lost their pedestrian and have "drifted away".

In our framework, we use the Euclidean distance between the centre position of the tracker and the centre position of the detection. The distance is used as a cost measurement, so the higher the cost, the smaller the chance the tracker and detection belong together. When the pedestrians in the scene are far apart, we can use the closest detection (within a certain distance) as best match. When pedestrians are close together, the assignment problem, where for each detection at most one tracker can be matched, becomes $O(B! - (B - A)!)$ in size using a brute force approach to find an optimal solution, where B is the maximum of the amount of trackers and detections and A is the minimum. In [61], the *Hungarian method* is described that solves the assignment problem based on a cost matrix (a matrix indicating for each assignment the cost, the distance between the two in this case). Hereby the assignment between the detections on a frame and the running trackers is determined, such that the sum of the costs is minimal. The C++ library Dlib [57] contains an implementation of this algorithm that runs in $O(n^3)$. Although the assignment problem is not large enough to encounter real profit for such an efficient implementation, it results in an optimal assignment between the running trackers and detections. Note that the distance we use as a cost measurement here has no information about the identity of pedestrians but the location. When unique identification is required, the use of other metrics (e.g. colour, texture, ...) may be more optimal.

To reduce the searching space, and thereby the detection speed, we exploit the temporal information we obtain from the tracker. To initialise tracks, we run the tracker only in a limited area of the image where we can expect pedestrians to enter the frame, which we name *Initialisation Regions*. When initialised, the tracker gives us a thoughtful guess of the next position of the pedestrian in the prediction step. This prediction is used to crop a region from the image which, when the prediction is sufficiently accurate, will include the tracked pedestrian. By running the pedestrian detector on this limited area, we obtain a new measurement that can be used in the correction step of the tracker. To improve the chance of obtaining a detection, we run the detector at a low detection threshold, but only use the highest scoring detection. The search space is now reduced to only the *Initialisation Regions* and the regions around the predicted locations of the trackers. To reduce the search space even further, we extend the state vector of the Kalman tracker we gave in equation 5.1 with scale information. It now becomes:

$$x_k = \begin{bmatrix} x & y & v_x & v_y & s & v_s \end{bmatrix}^T \tag{5.3}$$

where s represents the scale factor between the original height and the current height of the pedestrian, and v_s represents how fast the scale changes between frames.

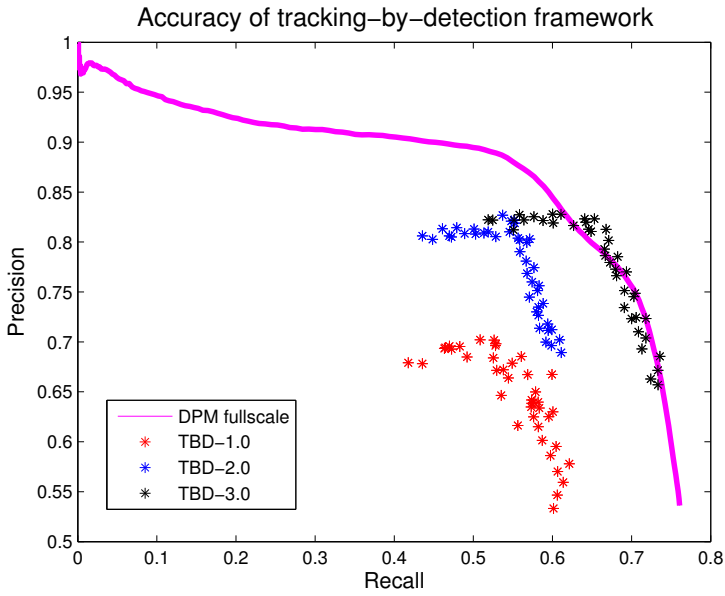


Figure 5.5: Comparison of accuracy using our Tracking-by-Detection framework.

The transition matrix now becomes:

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

Now we have on top of the area reduction also a scale reduction per tracked area. Figure 5.5 shows the accuracy of our tracking-by-detection framework with different parameters. For our experiments we use *Ours-DPM* as the pedestrian detection algorithm. As a first attempt we run *TBD-1.0*, where the lifetime is set on 25 frames, and we use three *Initialisation Regions*, one at the top of the frame, one at the bottom and one at the left, as shown in figure 5.6.

We can observe that both the Recall and the Precision are lower than the full-scale (where the detector is evaluated on each location and at each scale, as explained in section 2.2) evaluation. The main cause of the lower precision is

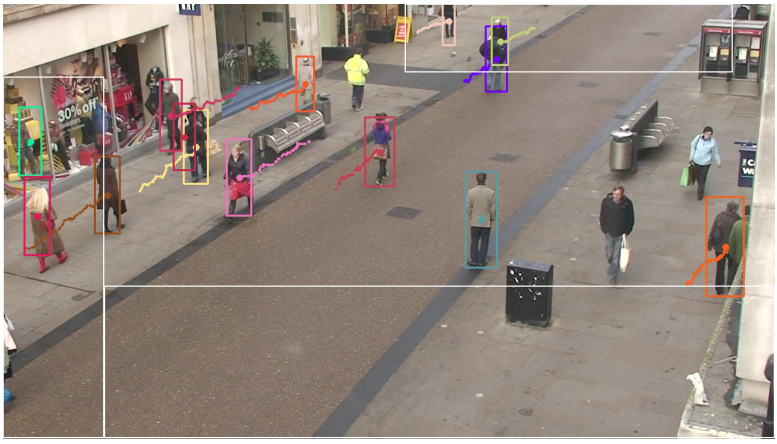


Figure 5.6: Tracking-by-detection using 3 *Initialisation regions*, as used by TBD-1.0 and TBD-2.0.

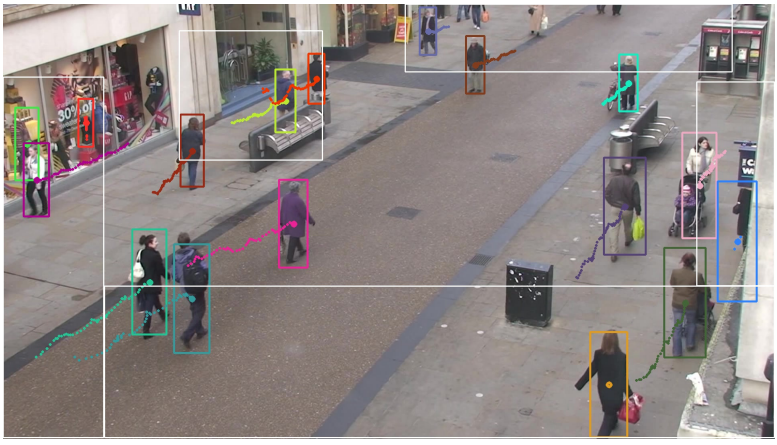


Figure 5.7: Tracking-by-detection using 5 *Pickup regions*, as used by TBD-3.0

that tracks are kept too long "alive", even after pedestrians have left the frame and when the tracker has drifted away, which increases the amount of false detections. When we lower the lifetime to 10 frames, we obtain the accuracy of *TBD-2.0*, where we can clearly observe the improved precision-value. To increase the recall value, which represents the amount of annotations we actually find, we just have to increase the area we search for pedestrians on. After a visual evaluation of *TBD-2.0*, we found out that in the region at the right of the image, and the region around the bench, much occlusion occurs, which leads to lost tracks (and thus missed detections). Adding these regions as *Initialisation regions*, as shown in figure 5.7, obtains the accuracy of *TBD-3.0*, and match the detection accuracy of the full scale processing of *Ours-DPM*. Note that further parameter-tuning can further improve the detection accuracy.

Currently determining the parameters, the lifetime and the *Initialisation Regions*, of this tracking by detections framework are made based on intuition and visual inspection. In our future work, section 8.1, we show a possibility on how this process can be automated to avoid this manual case-specific optimization.

In table 5.2, we compare the processing speed of our tracking-by-detection approach with a full scale processing of *Ours-DPM*. As we can observe, we obtain a high speed improvement with the proposed search space reduction, although it requires some parameter tuning, which depends both on the application and the used detection algorithm. Also note that the speed of a tracking-by-detection framework as we described is dependent on the amount of active trackers, and thus pedestrians in the scene. If no tracks are running, the speed is equal to the evaluation of the *Initialisation regions*, while each additional tracker leads to an extra region to evaluate and so an additional computational cost.

Technique	Frame rate	Speed-up
Ours-DPM	0.237 fps	1.0x
TBD-1.0	0.8862 fps	3.74x
TBD-2.0	1.055 fps	4.45x
TBD-3.0	0.73811 fps	3.11x

Table 5.2: Comparison of the detection speed when using our tracking-by-detection framework.

5.5 Exploiting the ground plane constraint

5.5.1 Introduction

In section 5.3 we showed how the reduction of the number of scales helps to improve detection speed, with no impact on accuracy. In section 5.4 we improved speed drastically by integrating temporal information in the form of a tracking-by-detection framework. The search space was reduced to only the locations where we expect pedestrians to be and only scales around the predicted pedestrian's height are searched for.

In this section we integrate an estimate of the ground plane to reduce the search space. When a pedestrian is further away from the camera, he will appear smaller in the image. Since we could expect pedestrians standing on the ground, we can find a relation between a point on an estimated ground plane (a y-coordinate), and the expected height of the pedestrian in pixels. It is this relation we will exploit here.

5.5.2 Approach

Ground plane estimation is the technique to approximate the orientation of the ground plane in 3D relative to the position of the camera. This allows approximating the height of an object for each position in the image and thus enables a restriction on the search space. In [84] is described how the object height in an image can be predicted based on the homography of the ground plane and the real world size of the object to detect. Based on the desired variance on the real world size, e.g. detecting pedestrians between 160cm and 185cm, rectangular regions are cut from the image and transformed to a standard size, where each region should be evaluated at the same scale. They conclude that the relation between the pedestrian height and a y-position in the image, can be approximated as a first order linear function for a horizontally mounted camera (yaw angle of 0°).

Since we do not have 3D information such as a ground plane homography, and so cannot determine the real orientation of the ground plane relative to the camera, we make an estimation based on the available annotations. Figure 5.8 shows our approximation for the annotations on the Town Centre dataset in black. As can be observed, a lot of annotations do not comply exactly with this function. According to our publication "On-board real-time tracking of pedestrians on a UAV" [22], the two main causes are: the variation in object height and the tilt movement of the camera relative to the ground plane. Since

the camera in the Town Centre dataset is fixed, we only compensate for the variation in pedestrian height, as shown in figure 5.8.

We performed the same task for the Caltech dataset and the ETH dataset. In these datasets the camera is mounted on a driving car and a moving stroller respectively, which causes the camera to move relative to the ground plane with both a parallel translation and a pitch rotation. This pitch rotation results in an additional deviation from our ground constraint function. When this movement is small, we can compensate for this by using a fixed pixel offset. In figure 5.9 and figure 5.10, we see the height compensation we did before on top of the tilt offset (dashed lines).

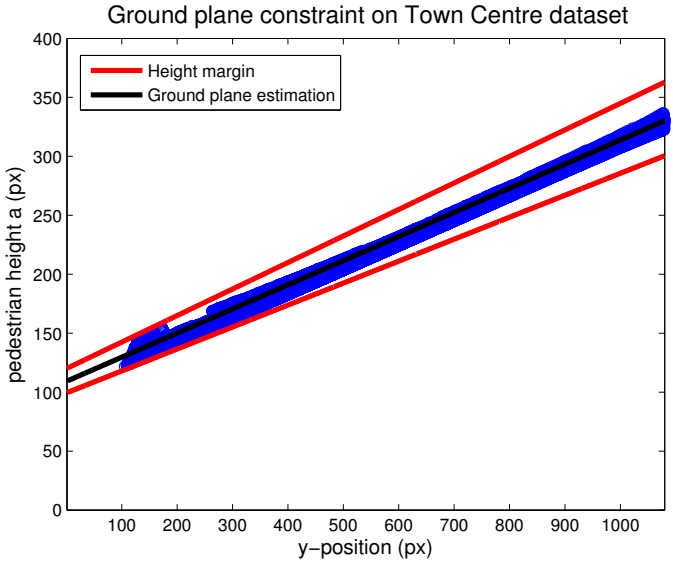
Now we can determine the image region each specific scale should be searched in. The boundaries of this region are defined by the intersection of the object height with the margin lines, which indicate the combination of both compensations (the height variation and/or camera movement). Keep in mind that these y-values determine the location on the ground plane, and so this region has to be extended with the object size we expect at that location. This is visualised in figure 5.11 for an object size of 150px.

The use of a ground plane constraint benefits both the detection speed, by reducing the search space, and the accuracy of the detection process, since false detections that do not fall within the search space for that scale are avoided. In section 5.5.3 we validate this.

5.5.3 Ground plane constraint validation

For our evaluation of the ground constraint, we use three datasets, and evaluate multiple detectors on them. Figure 5.12 visualises the accuracy of multiple detectors on the Town Centre dataset using a *Precision vs. Recall* curve. We obtain a consistent improvement in accuracy by using the ground constraint. The same result holds for the ETH dataset and the Caltech dataset, as shown in figure 5.13 and 5.14 respectively, using the constraints we discussed in section 5.5.2.

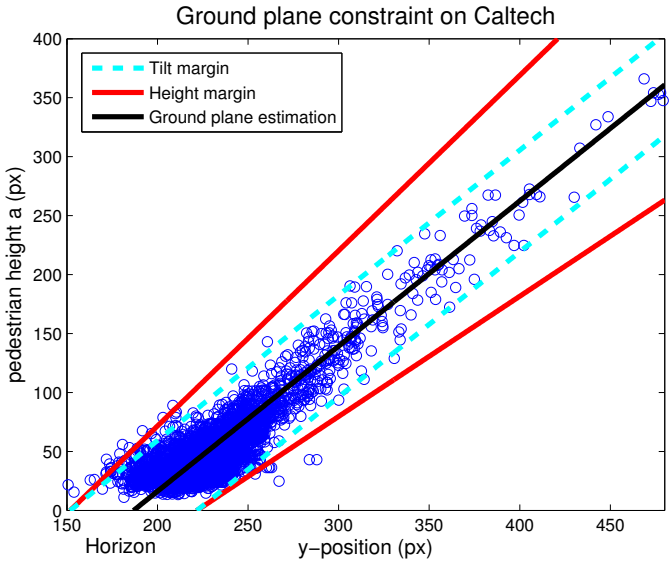
Note that the accuracy gain is obtained by the pruning of false detections that fall outside the cropped regions. Each point on an accuracy curve (*Precision vs. Recall* or *Miss rate vs. FPPI*) corresponds to the use of a certain threshold. Note that the use of the ground plane constraint will have no effect on the *Recall* or the *Miss rate*, such that when more detections need to be found, we still have to lower the detection threshold, possibly at the cost of processing speed.



(a) Ground plane constraint

(b) Example frame

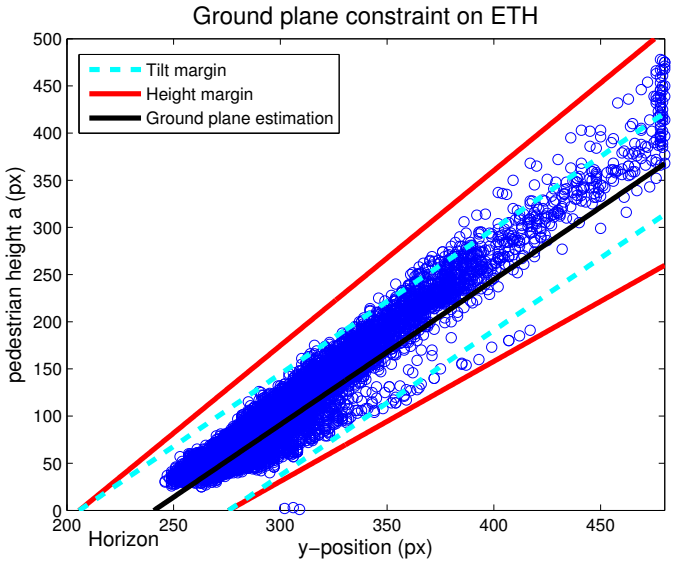
Figure 5.8: The ground plane estimation on the Town Centre dataset. The black line is our estimation function, while the other solid lines indicate the variance allowed on the scaling. The further to the right of the graph, the lower in the frames (higher y-position), and the larger the pedestrians.



(a) Ground plane estimation

(b) Example frame

Figure 5.9: The ground plane estimation on the Caltech dataset. The black line is our estimation function, while the other solid lines indicate the variance allowed on the scaling. The further to the right of the graph, the lower in the frames (higher y-position), and the larger the pedestrians.



(a) Ground plane estimation

(b) Example frame

Figure 5.10: The ground plane estimation on the ETH dataset. The black line is our estimation function, while the other solid lines indicate the variance allowed on the scaling. The further to the right of the graph, the lower in the frames (higher y-position), and the larger the pedestrians.

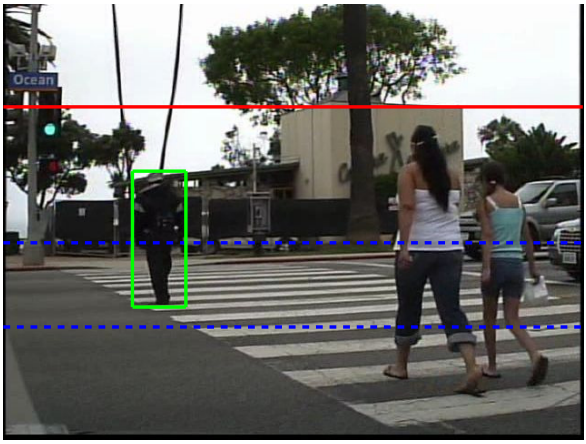


Figure 5.11: The region we crop to search on for a pedestrian size of 150px. The dashed lines indicate the boundaries on the ground plane, on top of this the expected object height has to be taken into account. An example detection is indicated with a bounding box.

We see that we obtain a considerable accuracy gain by using our ground plane constraint for most detectors, but it is not guaranteed. For the *Deformable Part model* detector on the ETH dataset for example, the accuracy gain is quite limited. This tells us that this detector has already most detections according to our estimated ground plane. The same holds for the expected processing time of the detectors (which we discuss later using table 5.3 and table 5.4). When most of the processing time is already spend according to the ground plane estimation, pruning parts of the search space will have less benefit for the processing speed.

In our experiments on the Caltech dataset, we added the results we obtained using an ACF-model we trained on the Caltech training dataset. These results where also published in our recent paper *"On-board real-time tracking of pedestrians on a UAV"* [22]. As a first observation, we can see a large accuracy gain by training on a dataset similar as the target dataset, a statement that is supported by literature [8, 94, 98].

As we described in chapter 2, the *Aggregate Channel Features* and *SqrtICF* implementation make use of feature approximation to obtain a higher processing speed. Instead of calculating all the layers of the feature pyramid from the

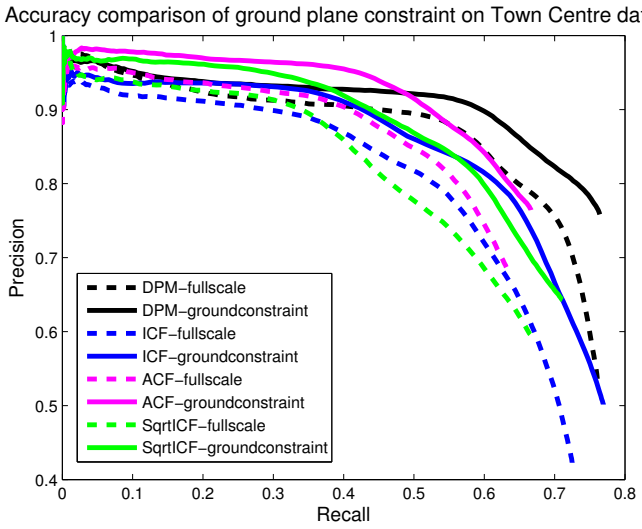


Figure 5.12: Accuracy improvement when using the ground plane constraint on the Town Centre dataset.

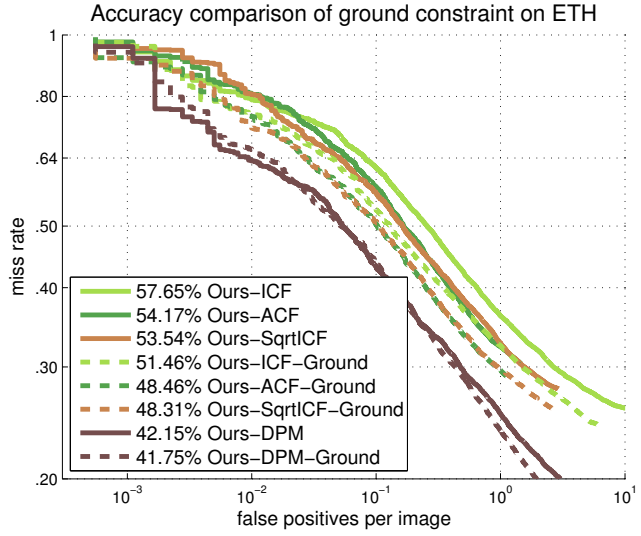


Figure 5.13: Accuracy improvement when using the ground plane constraint on the ETH dataset.

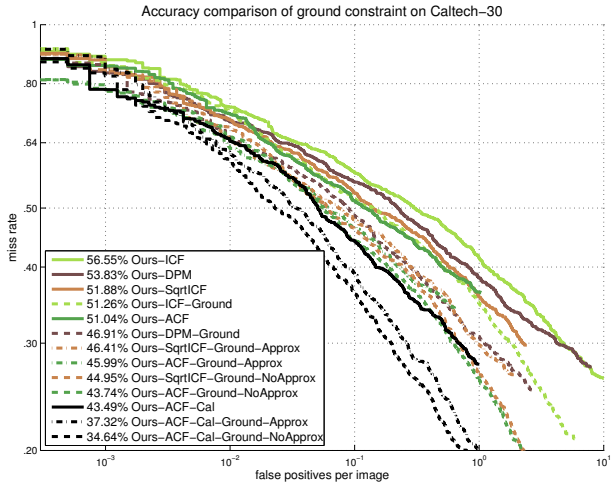


Figure 5.14: Accuracy improvement when using the ground plane constraint on the Caltech dataset.

corresponding image data in the scale-space layer, only a fraction (originally one layer per octave) of the layers is calculated this way, while the others are approximated to improve speed. For our ground plane experiments, we make a distinction between using this approximation or not.

To integrate feature approximation in combination with our ground plane constraint, we cluster a number of scales. For each cluster we determine the surrounding region. For this region we can fall back to the original method to calculate one time the real features per octave, and approximate the others from this layer. For our experiments, we cluster five scales, such that 80% of the feature layers can be approximated. Note that the larger the clusters, the less effective the ground plane constraint becomes in pruning the search space, but the more approximation we can apply. When applying this technique for an application, a balance needs to be made between these options. At one side the clusters contain only one layer, which means that no approximation is used, while on the other end all scales belong to the same cluster such that we have fallen back on full scale evaluation with no search space pruning.

In table 5.3 and table 5.4 we compare the detection speed of our implementations on the Town Centre dataset and the Caltech dataset respectively. As we can observe, the ground plane constraint brings a significant improvement in

evaluation speed considering the combined accuracy improvement. The choice of using approximation of features is dependent on the importance of speed versus accuracy. Note that the speed can be further improved by combining a tracking-by-detection framework with our ground plane constraint. This allows for each location, determined by a running tracker instance, to accurately determine the expected pedestrian's size. In the tracking-by-detection framework we described in section 5.4.2, the pedestrian's size is a tracked variable, but this is not always as accurately determined as when using a ground plane estimation. The more accurate the scale can be determined, the less scales need to be evaluated for each track, and so the less computation time is consumed. This combination is applied in the case study we describe in chapter 6, which uses the *Warping Window* approach we describe briefly in section 5.6.

Technique	Full scale	Ground constraint	Speed-up
Ours-DPM	0.237 fps	0.778 fps	3.28×
Ours-ICF	0.414 fps	1.27 fps	3.07×
Ours-ACF	1.12	2.33 fps	2.08×
Ours-SqrtICF	1.17 fps	2.27 fps	1.94×

Table 5.3: Comparison of the detection speed on the Town centre dataset when using the ground plane constraint.

Technique	Full scale	Ground constraint	Speed-up	Ground constraint + approximation	Speed-up
Ours-DPM	0.32 fps	0.95 fps	2.97×	/	/
Ours-ICF	0.68 fps	2.4 fps	3.53×	/	/
Ours-ACF	6.9 fps	8.2 fps	1.19×	11.6 fps	1.68×
Ours-ACF-Cal	8.1 fps	26.1 fps	3.22×	31.7 fps	3.91×
Ours-SqrtICF	6.78 fps	8.4 fps	1.24×	12 fps	1.77×

Table 5.4: Comparison of the detection speed on the Caltech dataset when using the ground plane constraint.

5.6 Warping Window

In this section we give a short overview of the *Warping Window* approach, since it is an extension of the ground plane constraint, and is mostly used in combination with a tracking-by-detection framework. In chapter 6 we will describe our hybrid GPU/CPU implementation we published in "Pedestrian detection at warp speed: Exceeding 500 detections per second" [25]. We used

Figure 5.15: Example frame taken with a truck's blind spot camera.

Figure 5.16: Example frame from a surveillance context.

this GPU/CPU implementation in combination with the *Warping Window* approach.

In the previous section we described how the integration of a ground plane estimation improves both the processing speed and the detection accuracy of pedestrian detection algorithms. However, we hypothesize that the decrease of the pedestrian's size is a first order linear function, and that for each y-position we can expect a similar pedestrian height. The use of wide-angle lenses, which are common, among other applications, in surveillance applications, results in a video stream that does no longer accord to these hypotheses. Wide angle lenses impose both a scale variation and a rotation of the object, as can be seen in figure 5.15 and figure 5.16.

In 2012, K. Van Beeck et al. [87] proposed a solution for this problem in his research on the detection of vulnerable road user in the blind spot zone of trucks, where they had to tackle the same problem. They state that the amount of rotation and scaling is only dependent on the position in the image. This allows making a calibration upfront which for each (x,y) coordinate represents a specific scale and rotation. This calibration is achieved by using 100 manually labelled pedestrians who are homogeneously spread over the total image region. Each of these annotations (ground truth labels) contains both scale and rotation for that pedestrian, and as such for that particular position. A second order polynomial function is fitted through these data points forming a separate calibration map for the scale and the rotation, as shown in figure 5.17, which can be used while the truck is driving in the assumption the ground plane will not change.

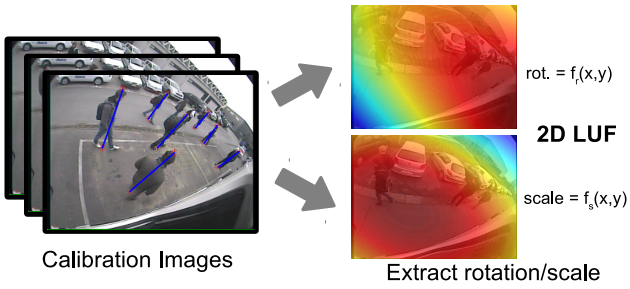


Figure 5.17: A one time calibration is needed, yielding the LUF for both the rotation and scale [87]

The warping window is commonly combined with a tracking-by-detection framework as we described in section 5.4. To validate the presence of a pedestrian, a *Region of Interest* (ROI) I is cropped from the image according to rotation and scale at that location.

Based on the calibrated values, this ROI can now be warped such that the pedestrians becomes upright at a fixed scale using $I_{warp} = TI$, with transformation matrix T:

$$T = \begin{bmatrix} s\cos\theta & -s\sin\theta & t_x \\ s\sin\theta & s\cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

This warping approach ensures a straight up model such that the pedestrian approximates the model height, which is 120px for the DPM detector used here. This allows reducing the search space to limited locations and at a single scale. The resulting detection is than retransformed to the original image and fed into the tracking framework. This process is visualised in figure 5.18. More details can be found in [87] which more deeply describes this procedure.

In the scope of our publication "Pedestrian detection at Warp Speed: Exceeding 500 detections per second" (EVW 2013) [25] we applied the Warping Window approach on the Caltech dataset. The basic viewpoint of this dataset allows to quantitatively show the accuracy improvement of the Warping Window approach over a classic full scale evaluation. Note that here we use the warping window as a post-processing step, whereas it is normally used in combination with a tracking-by-detection framework as a pre-processing step to reduce the search space.

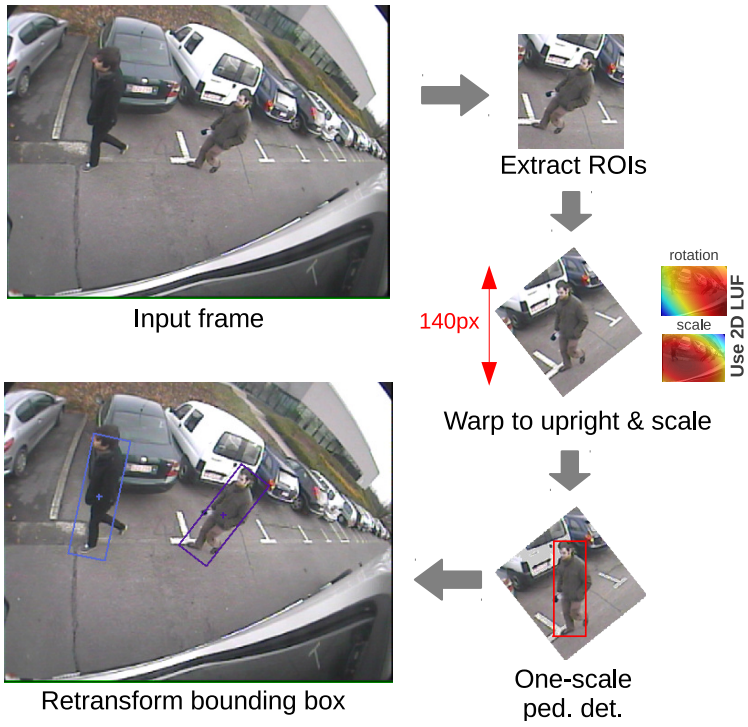
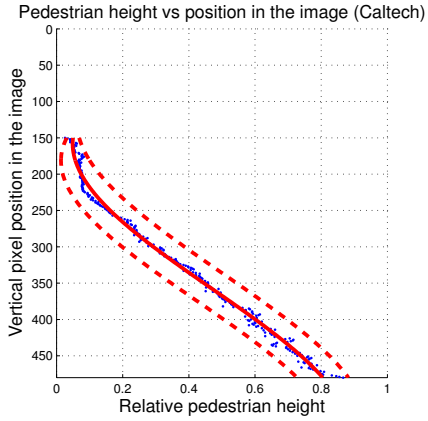


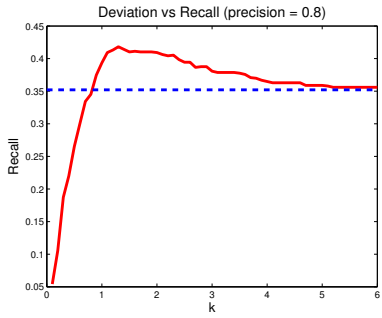
Figure 5.18: Warping Window approach.

Due to the camera viewpoint in the Caltech dataset, no rotation is needed, hence we only model the pedestrian's scale at each image position. Also here we assume that at each horizontal pixel line the scale is constant, reducing the dimensionality of the function to one. To make our calibration, we calculate the average height in pixels of the ground truth for each horizontal pixel line. Annotations above the horizon are discarded. These data points are visualized in figure 5.19a. Next we fit a third order polynomial function through these data points (solid line). The dotted lines illustrate two times the standard deviation (2σ) at each horizontal position. This transformation model can then be used to prune the results and show the potential accuracy improvement.

If the height of a new detection is *inconsistent* with what is expected at that particular position, the detection is discarded. As a measure of inconsistency, we use the degree of deviation. If we allow much deviation, no or only slight improvements are obtained since little pruning is applied, while on the other hand limiting the possible deviation too much leads to a significant drop in



(a) Pedestrian height vs. y-position.



(b) Influence of k-value on recall rate.

Figure 5.19: The warping window approach applied to the Caltech dataset

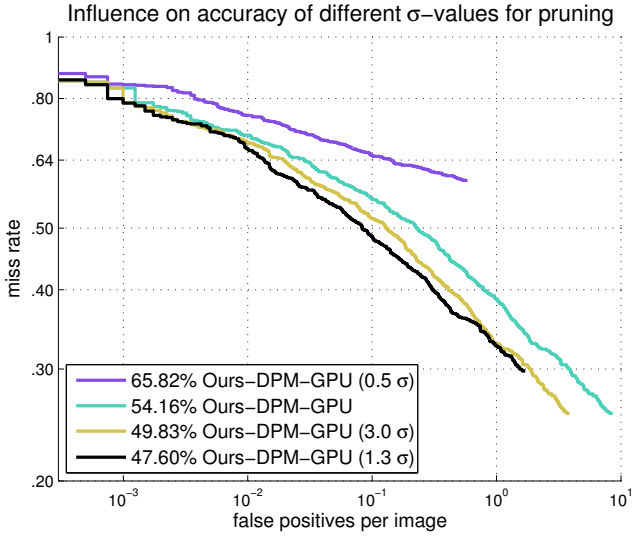


Figure 5.20: Accuracy improvement achieved on the Caltech dataset when using the warping window approach

the recall rate. We empirically determine the maximum allowed deviation (expressed as $k\sigma$) and evaluated the recall rate at a constant precision rate (80%). These results are displayed in figure 5.19b. Here one clearly sees that the optimal deviation is found around 1.3σ, while at higher values of k the recall rate converges to the recall rate of the reference implementation (displayed as the dotted line).

Note that in the approach of section 5.5.2, the pruning was performed as a pre-processing step, such that the detector is not evaluated on locations that do not cope with the ground constraint, which allows an additional speed-up on top of the accuracy improvement.

Figure 5.20 gives the miss rate versus the FPPI for both the original implementation along with three values of the deviation; an optimum is reached at 1.3σ. Applying the warping window approach thus leads to an accuracy improvement: at e.g. 0.1 FPPI, the miss rate decreases from 55% to 48%. Note that although we use a third order polynomial to model the ground constraint here, exploiting the ground constraint as a first order linear function, as we described in section 5.5.2, reaches an equal accuracy improvement on this dataset.

Note that the Warping Window approach has some deviations from the tracking-by-detection framework and ground plane constraint we described before. The tracking state does not include scale information, since this information is calibrated up front, and so is determined from the location in the image. There is also no margin used to compensate for a moving camera and/or variations in pedestrians height. The DPM detector that is used for this approach in the original paper allows a small deformations. As long as the deformations from the pre-calibrated scale and rotation are sufficiently small, the DPM detector is sufficiently robust to avoid the requirement of these margins.

The accuracy of this approach obtained on a dedicated blind spot experiment, according to [87], is visualised in figure 5.21. This accuracy is obtained on a test set of about 1000 pedestrians in very diverse poses and movements. In this dissertation we do *not* alter this approach, such that the same results could be expected. A TP occurs when the centroid of the annotated pedestrian lies in a circular region with a radius of 25% of the calibrated height at the position of the detection's centroid.

Note that this warping window approach is much more flexible than standard ground plane assumptions. This approach is easily generalizable to complex arbitrary camera viewpoints, and situations where non-linear camera distortion and extreme viewing angles occur (e.g. surveillance cameras, wide-angle lenses and so on).

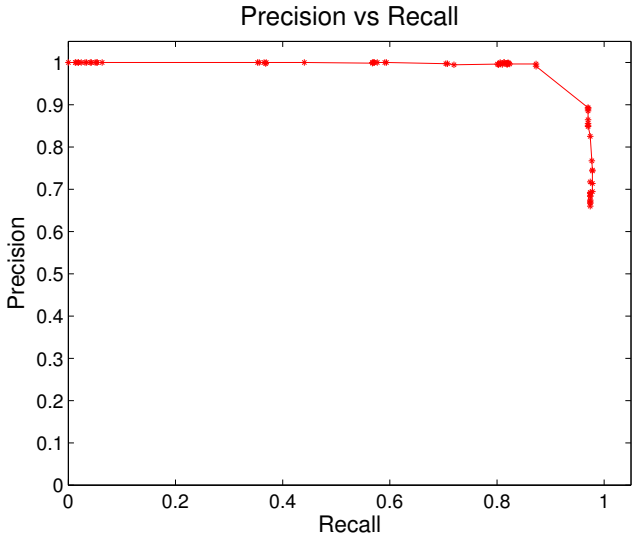


Figure 5.21: Accuracy of the Warping Window approach on a dedicated blind spot experiment. Taken from [87].

We will apply this method to improve the detection speed in the first case study in chapter 6 We will apply the *Warping Window* approach in the application it was designed for: the detection of pedestrians in the blind spot of a truck.

5.7 Conclusion

In this chapter we discussed techniques to improve the detection speed by reducing the search space of the detection task, by integrating scene knowledge. The techniques we described here are complementary with the techniques we described in chapter 4, where multi-core architectures where used to improve the detection speed. The two main approaches we described in this chapter are the use of a tracking-by-detection framework and the ground plane constraint as a first order linear function.

In our tracking-by-detection framework, only a limited area of the images is evaluated for each frame. This allows initialising a tracker for each pedestrian entering the scene. As long as the tracker is alive, a pedestrian is searched for around the predicted location of the tracker. The speed benefit we obtain is dependent on the amount of pedestrians in the scene, since this determines

the amount of running tracks. We showed that by optimising this approach for the scene, we can reach the same accuracy as with evaluating the complete frames while being more as 3 times faster while using the crowded *Town Centre* dataset. Since this optimisation is dependent on the pedestrian detector in use, we only demonstrated this approach using the *Deformable Part Model* detector.

We demonstrated how a ground plane estimate as a first order linear function can improve the detection speed drastically. Since ground plane estimation allows to approximate the expected pedestrian size at each location, it allows to divide the frames into regions which are each evaluated on a specific scale. The boundaries of these regions are determined by the use of a margin to compensate for the variation in height of pedestrians, and to compensate for the movement of the camera. We demonstrated the benefit for both detection speed and accuracy improvement of the ground plane estimation approach using the Caltech, ETH and Town Centre dataset.

We introduced the *Warping Window* approach, since it is closely related to the ground plane estimation approach we presented in this chapter, although it allows to cope with non-linear camera distortion due to wide angle lenses.

Chapter 6

Application I: Pedestrian detection in a truck's blind spot camera

In chapter 2 we introduced how pedestrian detection algorithms work, and presented a framework containing implementations of different pedestrian detection algorithms. In chapter 4 and chapter 5 we presented techniques to improve the processing speed of these algorithms.

In this chapter, we focus on the application of the detection of pedestrians in the blind spot zone of trucks. This is a challenging application, since it requires high accuracy to avoid false alarms while detecting all pedestrians, in combination with a strict time constraint. We created a hybrid GPU/CPU implementation of the *Deformable Part Model* detector, which in combination with the *Warping Window* approach we described in section 5.6, exceeds 500 detections per second. This work is published as "Pedestrian detection at Warp Speed: Exceeding 500 detections per second" (EVW 2013) [25]. Note however that the use of this approach is not restricted to the blind spot application, but can easily be applied on any application that uses a tracking-by-detection approach. The accuracy results we show in this section are updated according to our most recent implementations, which we discussed in section 4.7.

The paper "Pedestrian detection at Warp Speed: Exceeding 500 detections per second" [25] was co-authored by K. Van Beeck. Combining a hybrid GPU/CPU implementation of pedestrian detection to exploit full hardware capability, with a tracking-by-detection framework, was proposed by Floris De Smedt. The

work of K. Van Beeck on the detection of vulnerable road users in the blind spot area of trucks formed a good application for this approach. The required technique of the Warping Window approach, and showing its potential on the Caltech dataset [87] was contributed by K. Van Beeck.

6.1 Introduction

A pedestrian detector that is at the same time fast and accurate, would open up a wide variety of applications, including robotics, surveillance and automotive safety. These applications clearly benefit from the high robustness most recent algorithms can achieve. Indeed, over the past few years impressive accuracy improvements were obtained on challenging benchmark datasets, containing a wide variety of poses and appearances. Unfortunately, high accuracy often comes at the cost of high computation time, making these algorithms infeasible in real-life applications. We overcome this problem by optimizing an already accurate detection algorithm using algorithmic optimization and the exploitation of scene constraints.

We present an efficient pipelined hybrid CPU/GPU pedestrian detector implementation. This implementation exploits both the parallelisation opportunities of multi-core CPU and GPU (chapter 4). While being fast on its own, we further improve the speed-up using the *warping window approach* in combination with a tracking-by-detection framework, to cope with the non-linear camera distortion (as we described in section 5.6). Hereby we combine both the speed improvements we described in chapter 4 and chapter 5. We benchmark our hybrid pedestrian detector implementation both with respect to accuracy as speed on the Caltech dataset [33, 34], and show that despite the speed-up, we achieve state-of-the-art accuracy.

Finally, we propose experiments and results on a second dataset targeting a demanding application: the detection of pedestrians in the blind spot camera images of a truck. Using this application we illustrate the full potential of this warping window approach, and achieve excellent accuracy results with very high processing speeds (500 detections per second). Such speeds are useful for e.g. crowded scenes or when implementing on embedded hardware with limited processing power. The remainder of this chapter is structured as follows. In section 6.2 we describe our hybrid CPU/GPU pedestrian detector, while in section 6.3 we discuss the warping window approach with quantitative accuracy improvement results. We then combine both approaches in section 6.4, and show how our speed-up is realized. We conclude this work in section 6.5.

6.2 Hybrid Pedestrian Detector

Speed improvements can be obtained in two ways: either optimize the execution speed (as we described in chapter 4) or decrease the search space (as we described in chapter 5), or a combination of both. In this section we discuss the first, while in section 6.3 we target the latter. The accuracy and speed results discussed in this section therefore are measured without scene constraints, thus applying a full scale-space search. The object detection algorithm we start from, is the *cascade latent SVM detector* proposed by Felzenszwalb et al. [42], which is the *Deformable Part Model* detector we use in the previous chapters. As we already have demonstrated, this detector achieves state-of-the-art accuracy results. An additional advantage of this detector is the allowed pose variation to cope with both (small) perspective transformations of the pedestrians due to bird eye view, and the errors in the calibration. We already discussed this detector in detail in section 2.3.

Using this original part-based pedestrian detector Matlab implementation [40] as a baseline, we propose two implementations to improve the detection speed. In the first implementation we ported the calculation of the feature pyramid to the GPU (as discussed in section 4.7), resulting in a significant speed-up. Our second implementation consists of a multi-threaded hybrid CPU/GPU implementation, combining the GPU implementation with the pipeline implementation we introduced in section 4.6, achieving a speed-up of a factor of 12.7 over the original implementation. In section 6.4 we then integrate this last implementation with scene constraints and temporal information, and propose our *WSPD* (*Warp Speed Pedestrian Detector*), which achieves pedestrian detection at 500 detections per second.

For the remainder of this section we discuss our first implementation (feature pyramid on GPU), the multi-threaded hybrid CPU/GPU implementation, and give evaluation results concerning both speed and accuracy.

6.2.1 GPU Feature pyramid implementation

The feature pyramid consists of multiple layers, each containing the features of a rescaled version of the source image. The publicly available implementation (referred further as *LatSVMV4 (original)* - referring to the latent variables) does not suffice for real-time applications. A first improvement in speed is obtained by porting this Matlab implementation to a full C++ implementation, which we described in section 2.5. We will use this implementation, under the alias *WSPD-v0.1 (float)* for speed and accuracy comparisons in this chapter. For maximum speed benefit of the Nvidia GPU hardware, available in our hardware

setup, we ported the OpenCL implementation we described in section 4.7 to a CUDA implementation, allowing an extra 25% speed-up. In essence CUDA is a C extension that allows the use of Nvidia GPU hardware as an execution device, enabling faster execution of algorithms that use data parallelism. We further refer to this GPU implementation as *WSPD-v0.2 (GPU)*.

6.2.2 Multi-threaded hybrid implementation

Although the use of GPU hardware allows for a significant speed-up, it does not fully exploit the capabilities of the hardware system, since the CPU is only active when the GPU is idle and vice versa. To further speed-up the algorithm, and to circumvent this problem, we propose an implementation using a pipelined object detection scheme. In particular, we calculate the feature pyramid of a frame (on GPU) while at the same time performing the model evaluation step for each layer in the feature pyramid from the previous frame (on CPU). This way the CPU and GPU are both active as a hybrid system, allowing an increased detection throughput. Besides running the feature pyramid and model evaluation in parallel on CPU and GPU, we can further increase the detection throughput by matching the execution time of each step in the pipeline. The feature pyramid e.g. is almost twice as fast as the model evaluation step, so for each feature pyramid process we run two model evaluation processes, leading to a higher speed. An even faster detection throughput is achieved when running multiple instances of the detection pipeline in parallel. Figure 6.1 shows a schematic overview of our implementation structure, which we further refer to as *WSPD-v0.3 (hybrid)*. This structure focuses on the use of eight threads (one per block). As shown, we use two detection pipelines in parallel. We will discuss each block of the schematic overview in more detail below. Note that we combine data-parallelism, which is used to calculate multiple feature pyramids and model evaluations at the same time (horizontal blocks), and task-parallelism, since our pipelined implementations allows to run the different tasks in parallel (vertical blocks).

Pre-processing Here, all pre-processing (e.g. cropping, rescaling, rotating and search space pruning) is performed. The results are placed in the image queue for further processing.

Feature Pyramid In this block the feature pyramid is calculated. The images are gathered from the image queue, while the feature pyramids are pushed into the pyramid queues. Since the calculation of the feature pyramids is executed on GPU, the data transfer to the GPU is also included here. In the schematic overview of fig. 6.1, two instances of the feature pyramid are displayed,

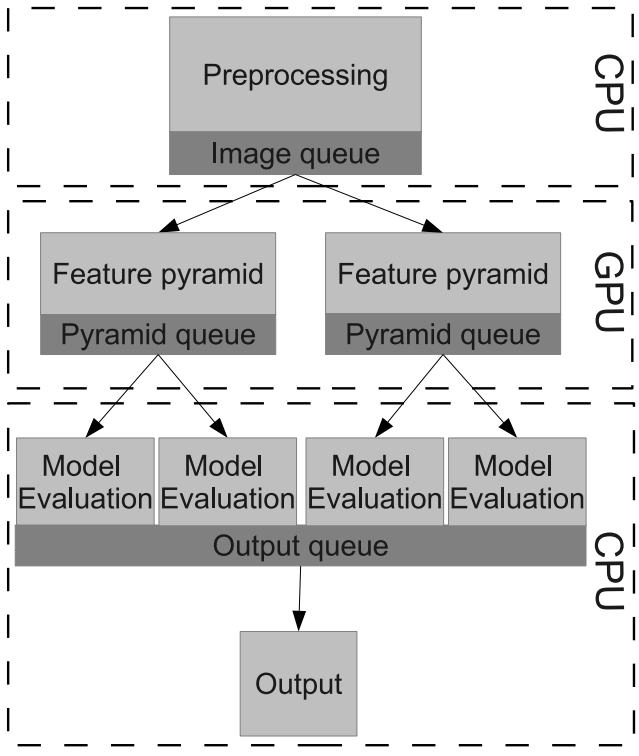


Figure 6.1: A schematic overview of the hybrid detector.

indicating that the feature pyramid of two frames are computed in parallel. Due to the modular approach of our innovative hybrid detecting scheme the implementation is easily hardware scalable. In our experiments each feature pyramid instance ran on a separate CUDA device (our GPU includes two such devices). Although it is possible to run multiple instances on the same CUDA device, evidently this is limited by the GPU resources.

Model Evaluation For each calculated feature pyramid, we need to evaluate the pre-trained model on each layer in the pyramid. Each feature pyramid feeds two model evaluation processes: this is needed since the feature pyramid calculation is almost twice as fast as the model evaluation. Since the feature pyramids are independent of the model to be detected, each instance of the model evaluation block is able to search for another model, thereby reducing calculating time if one aims to detect multiple object classes at once.

	640 × 480	1280 × 960	Speed-up
LatSVMV4 (orig.)	1.33 fps	0.33 fps	1×
WSPD-v0.1 (float)	1.6 fps	0.37 fps	1.12×
WSPD-v0.2 (GPU)	2.97 fps	0.85 fps	2.58×
WSPD-v0.3 (hybrid)	12.9 fps	4.17 fps	12.7×
fastHOG	10.6 fps	2.67 fps	

Table 6.1: Speed results of our warp speed pedestrian detectors compared to the public implementation of the algorithm and *fastHOG*. No scene constraints are applied yet

Output This final block gathers all the processed results and performs the post processing such as NMS (*Non-Maximum-Suppression*), reordering the frames, displaying detections and saving the frames.

6.2.3 Evaluation

We performed thorough experiments concerning both speed and accuracy of our implementations. Note that, at this point, all experiments are still performed without scene constraints, which we introduce in section 6.3. The speed comparison of our and publicly available implementations is given in table 6.1. All speed measurements were obtained on the same hardware (Intel Core i7 CPU 965 @ 3.20GHz with 12GB RAM and one Nvidia GTX295 GPU). Our experiments were performed on both the original image size (640 × 480) as well as on an up-scaled version (1280 × 960). This up-scaling is required when detections smaller then the model need to be detected. The part-based HOG-model is trained to detect pedestrians with a pixel size of 120px. We compared the relative speed-up we obtained to the original algorithm and fastHOG [78], a publicly available fast implementation of the HOG algorithm [19]. As can be seen in the table, our C++ reimplementation with *floats* is slightly faster than the original implementation. With our implementation of the feature pyramid on GPU we achieve a speed-up of a factor 2.58. While already faster, the overall speed-up is limited since we only implemented the feature pyramid on GPU. Our multi-threaded hybrid CPU/GPU implementation achieves a speed-up of a factor 12.7, thereby allowing real-time processing of 640 × 480 images (12.9 fps). Evidently the accuracy of the original detector should remain when using our hybrid implementation to improve speed. Therefore we compared our implementations of the algorithm with existing algorithms on the publicly available Caltech dataset [33]. All experiments are performed on the challenging

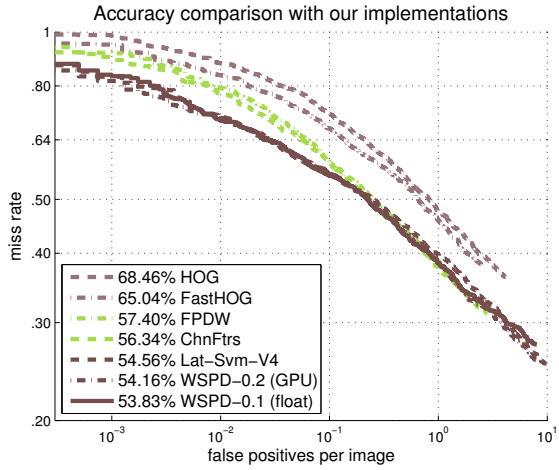


Figure 6.2: Comparison of existing detectors and our implementation. The results of HOG [19] and FPDW [30] are obtained from [34]

reasonable experiment setting. Our results are visualized in figure 6.2, where we display the miss rate versus the false positives per image (FPPI). The part-based detector (*LatSVMV4* [42]) achieves excellent accuracy results as compared to other state-of-the-art detectors (e.g. *FPDW* [30], *ICF* [32]), combined with the allowed pose variation which is especially beneficial in applications such as ours. Concerning our implementations, our speed-up does clearly not come at the cost of a performance drop.

6.3 Applying scene constraints

To improve the speed of our hybrid implementation even further, we reduce the search space using the *Warping Window* approach, described in section 5.6, as a ground constraint in combination with a Kalman based tracking-by-detection framework. As we have demonstrated on the Caltech dataset, has the *Warping Window* approach the same accuracy improvement as the ground constraint we used in section 5.5, but on top of that it can handle more complex scenarios, e.g. extreme camera viewpoints and wide-angle lens distortions, which are present in the blind spot application.

6.4 Warp Speed Pedestrian Detector

In this section we now present the combination of the previously discussed warping window approach (section 5.6) and the multi-threaded hybrid CPU/GPU implementation (section 6.2). As mentioned, this implementation achieves 12.9 fps without the use of scene constraints. Here we demonstrate the integration of the warping window approach and propose our *Warp Speed Pedestrian Detector* (WSPD) which achieves pedestrian detection at up to 500 detections per second, without loss in accuracy. We illustrate its potential on a challenging real-life problem: detecting pedestrians in the blind spot zone of a truck.

6.4.1 Single ROI selection

In section 5.6 we described how the warping window approach is used to reduce the search space. At each pixel location only one scale needs to be evaluated. To further reduce the search space, we can use techniques that limit the number of locations where a detection needs to be performed, based on some detection probability measure. In a fixed scene, simple and efficient background modelling techniques (e.g. background subtraction) can be used. However, in moving scenes with a highly dynamical background more complex techniques are necessary. One such technique is pedestrian tracking. If we are able to track the pedestrians throughout the scene, we can predict the search location (called tracking-by-detection), thereby reducing calculation time. Moreover we observe that in most applications the position where pedestrians enter the scene is predictable (e.g. doors). In the demonstrated blind spot application (discussed below), the Kalman tracker [55] has proven to be a robust technique for this purpose [87]. It predicts future positions based on detections in the past combined with a specific motion model. Based on this prediction, pedestrian ROIs are determined. For more detailed information on this we refer to [87].

6.4.2 Blind spot application

In a lot of real-life computer vision applications, due to the specific position of the camera, distorted camera viewpoints occur. Due to its flexibility, the warping window approach can cope with this specific distortion. To show its full potential we use an application with a challenging camera viewpoint: detection of pedestrians in blind-spot camera images. Because of the 115° wide-angle lens a specific distortion pattern occurs introducing non-linear camera distortion (see figure 5.15 for an example frame). Since the camera is mounted on a

moving truck, we have to deal with a highly dynamic background. The LUFs (as described in section 5.6) are obtained by fitting 2D second order polynomial functions through sparse data points obtained during a one-time calibration step (fig. 5.17).

6.4.3 Pedestrian detection at Warp Speed

The crux of the matter is that we can use the warping window approach in combination with a Kalman tracker, and use the predicted search locations as input for our hybrid CPU/GPU pedestrian detector (section 6.2.2). This way we are able to perform pedestrian detection at unprecedented high speeds. Since we reduced the search space to a single scale and a single ROI, our detector only focuses on image content with high probability of containing pedestrians at a fixed scale, thus being very fast. The speed of our detector evidently depends on the size of the ROI(s) we extract from the source image. The speed we give in figure 6.3[1] uses a ROI of 140x75, as motivated in [87]. We display both the frame rate (fps) and the number of pedestrian detections per second (Hz) we can achieve with respect to the number of pedestrians in the image. Initially with only one pedestrian in the image we achieve 480 fps, using the 140 pixels high ROI as motivated in section 6.3. If there are more pedestrians per image our pipeline is used more efficiently, thus the number of detections per second increases. However, this evidently leads to a decrease in the number of frames per second. At e.g. 20 pedestrians per image we still achieve an impressive frame-rate of 25 fps.

Using this technique, we are an order of magnitude faster than the Matlab implementation, which is optimized with parts in C-code using the mex API, described in [87].

6.5 Conclusion

In this chapter we applied multiple of the techniques we discussed in previous chapters for the application of detecting pedestrians in a truck's blind spot zone. To overcome the challenging requirements of this application, we presented a technique to evaluate the accurate *DPM* detector at high speed. To exploit the full hardware capability, we presented a hybrid GPU/CPU implementation which combines multi-core processing on CPU, with a GPU implementation of the feature pyramid of *DPM* (chapter 4). This optimisation obtained a speed-up of a factor 12.7 over the original Matlab implementation. To obtain an

[1] Both warping and detection times are taken into account here.

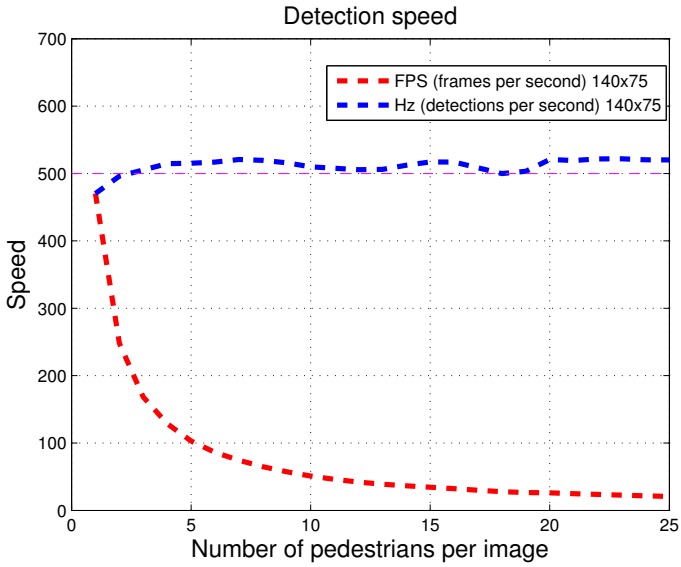


Figure 6.3: Speed results of our Warp Speed Pedestrian Detector in function of the amount of pedestrians to track.

even higher processing speed, we reduced the search space (chapter 5) using a tracking-by-detection framework based on the Kalman tracker and the Warping-Window approach, which is essentially a ground plane estimation approach capable of handling the non-linear camera distortion of the wide angle lens of the blind spot camera. This led to a frame rate dependent on the number of pedestrians in the scene. When the detection pipeline is full, we obtain a throughput of 500 detections per second, which means that even with 20 pedestrians in the scene we still obtain 25fps.

Chapter 7

Application II: On-board real-time tracking of pedestrians from a UAV

In this chapter we focus on the application of following pedestrians from a flying UAV (Unmanned Aerial Vehicle) while using only on-board processing. The main challenge of this application is the real-time evaluation of pedestrian detection and pedestrian tracking on an embedded system with limited computational power.

To cope with the limited computation resources of the on-board embedded system to perform pedestrian detection, we present a generalisation of the ground plane estimation approach we described in section 5.5 so it becomes parameterizable by the inertial sensors of the UAV. This work was published as "On-board real-time tracking of pedestrians on a UAV" [22] for which we received the best paper award at the *Embedded Vision Workshop* (EVW) 2015, which was held in conjunction with CVPR 2015.

This paper was co-authored by Dries Hulens, who took care of the interfacing with the inertial sensors of the UAV, using the Mavlink protocol, and the hardware selection. He also contributed the code for the software based PID-loop to steer the UAV and performed the configuration of its parameters. The implementation of the computer vision algorithms, the generalisation of the ground plane estimation to use the sensor data, and the parallel software design was contributed by Floris De Smedt.

7.1 Introduction

During the past years, UAVs have gained the attention for many applications, both for industrial and consumer use, such as aerial photography, surveying, cinematography, surveillance applications and rescue operations. Commonly the UAV is manually controlled using a remote control. In this chapter we focus on the application of automatically following pedestrians using a camera connected to an on-board embedded system, which is selected based on its computational power and weight criteria. To obtain an accurate pedestrian location in each frame, we combine a pedestrian detector with a particle tracker based on colour information. Since these algorithms are executed on an on-board embedded system, the results can directly be used to steer the UAV without human intervention. To improve both accuracy and processing speed of our pedestrian detection algorithm, we extend the commonly used ground plane estimation technique to a more generalised form based on the UAV sensor data, which is used as a search constraint. The technique described in this chapter can be used in a lot of applications such as: cinematography (automatic filming of e.g. extreme outdoor sports), automatic patrolling for intruder detection, following in surveillance applications and guardian angel or outdoor monitoring of elderly and children.

In this case study we propose three contributions to the state-of-the-art. Firstly, we select an optimal embedded system based on both the required computational power and weight criteria to execute our optimised state-of-the-art algorithms in real-time, on-board on a UAV. Secondly, we propose a reformulation of the ground plane assumption such that it elegantly takes into account constraints and is parameterizable with instantly measured altitude and rotation values of the UAV. Finally, by combining our optimised and constrained pedestrian detection algorithm with a colour based particle filter, we allow a continuous PID-controlled steering of a real drone, which we validated on real-life experiments.

The remainder of this chapter is structured as follows. In section 7.2 we discuss the current related work on an optimal technique for the selection of embedded hardware and UAV applications. In section 7.3 we describe the speed improvement of the ACF detector by extending the ground constraint to the 6 DoF we are dealing with when the video is taken from a UAV. In section 7.4 we describe the implementation of the particle filter we use. In section 7.6 we describe both the hardware and the software construction for our application. In section 7.7 we validate our complete system to automatically follow pedestrians on real-life experiments. Finally we conclude this chapter in section 7.8.

7.2 Related work

The related work on pedestrian detection and possible improvements are already studied in detail in previous chapters. In this chapter however, we want to perform this task on an embedded system, mounted on a flying UAV. In this section we discuss the additional future work on this topic.

Hulens et al. [53] designed a tool to determine the best-suited hardware platform for an algorithm at a specified evaluation speed. Next to that, the tool estimates the maximum possible activity of an embedded system, taking into account the battery and power consumption. The requirement of minimum weight and power usage hinders the use of GPU-platforms. Therefore we focus on CPU-based embedded systems and suitable algorithms in this section.

In [68], Naseer et al. described a system to follow pedestrians using a quadrocopter. They use two cameras, one for determining the 3D position of the UAV based on markers on the ceiling. The second camera is a depth camera, which is used to detect a person in 3D. The image from the depth camera is warped based on the calculated 3D position. Parts of the calculations are performed on a ground station. Recently Pestana et al. [76] proposed a system similar to ours where a UAV is used to track and follow objects of various kinds (pedestrians, cars, ...). They still use a wirelessly connected computer to do the necessary calculations to perform the steering of the UAV, and also require an online learning stage. In our system, the UAV is fully autonomous, since all processing is performed on an on-board embedded system using a single camera. Next to that we use a general applicable pedestrian model (although models for other objects could be used) such that we do not require a learning stage. Our system is not restricted to lab environments.

7.3 Pedestrian detection

Due to the limited computational power we have available on our embedded system, we use the *Aggregate Channel Feature* (ACF) detector [28]. As we described in section 2.3, this detector reaches very high detection speeds, even on single-core hardware, while maintaining high accuracy. This makes this detector an optimal choice for our application.

To improve the detection speed, we apply the ground constraint we described in section 5.5.2. However, previously we obtained the parameters of our linear ground constraint based from the annotations, and assumed the relative orientation between the camera and the ground plane will not alter too much.

This assumption will not hold from a flying UAV however. In section 7.3.1 we reformulate this calibration approach by parameterizing it with measured sensor data from the UAV, making a calibration step superfluous.

7.3.1 Using a ground plane constraint with 6 DoF

In our application we perform pedestrian detection on images taken from a flying UAV. The 6 degrees of freedom (DoF), visualised in figure 7.1, make it impossible to apply a pre-calibrated ground plane estimation function. Happily enough the ground plane function is only influenced by 3 DoF: rotation around the x- and y-axis (called the roll and pitch respectively) and translating along the z-axis (altitude). Translating along the x- and y-axis and rotating around the z-axis do not influence the ground plane estimation function, and so they can be ignored. The IMU of the autopilot measures the values corresponding to these DoF, enabling us to compensate for them and derive a linear ground plane constraint function.

We will first reformulate the ground plane constraint such that the UAV's altitude can be used as an input parameter. Figure 7.2 visualises the scenario of a flying UAV with a forward-looking camera based on the pinhole model. The further away the pedestrian is located, the smaller, and the closer to the horizon, it will appear in the image. Figure 7.3 shows the corresponding frame as captured by the camera on the drone. The y-position of the pedestrian's feet (position on the ground plane) can be calculated by:

$$y = \frac{h_{im}}{2} + \frac{a}{A}B \tag{7.1}$$

where h_{im} is the height (in pixels) of the image. Hereby the assumption of a flat ground plane is made, which imposes the y-position of the horizon in the middle of the image.

For this equation, we know all parameters up front except for the real-world flying height (B) of the UAV. By measuring the altitude of the UAV, we can obtain the parameters of the first order function and reuse the approach we described in section 5.5.2.

The *roll* of the UAV is easy to compensate for, since this only requires rotating the image with the inverse angle, making the horizon to lie horizontally again. The last degree of freedom we have to compensate for is the *pitch*, with the camera looking slightly upwards (or downwards) instead of straight forward.

Just as explained in section 5.5.2 we use two margins to compensate for the possible variation in the pedestrian height, and the movement of the camera.

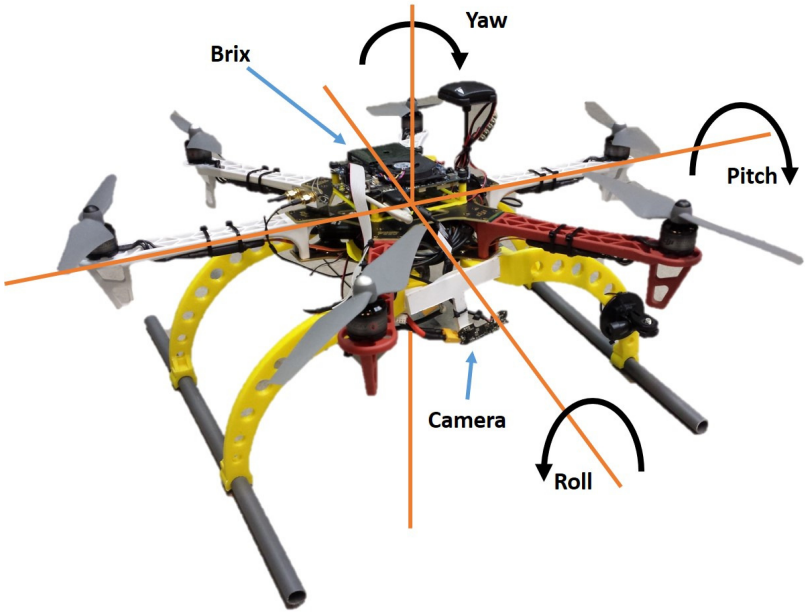


Figure 7.1: Our UAV system with the degrees of freedom shown, translating and rotating over the 3 axes.

The use of these margins is visualised in figure 5.9a. Note that since we will now compensate for the pitch using the inertial sensor data, this margin can be kept small to only compensate for small measurement errors. The equations of the solid lines, which represent the combination of these margins, now become:

$$a = y\frac{A_{\text{MIN}}}{B} - h_{\text{hor,MAX}} \tag{7.2}$$

$$a = y\frac{A_{\text{MAX}}}{B} - h_{\text{hor,MIN}} \tag{7.3}$$

where B is the measured altitude of the UAV, A_{MIN} and A_{MAX} the minimum and maximum real-world height of the pedestrians to be detected, and $h_{hor,MIN}$ and $h_{hor,MAX}$ the y-positions of the horizon in the image, deviating from the ideal $h_{hor} = \frac{h_{im}}{2}$ value due to pitch angle effects (see figure 5.9a).

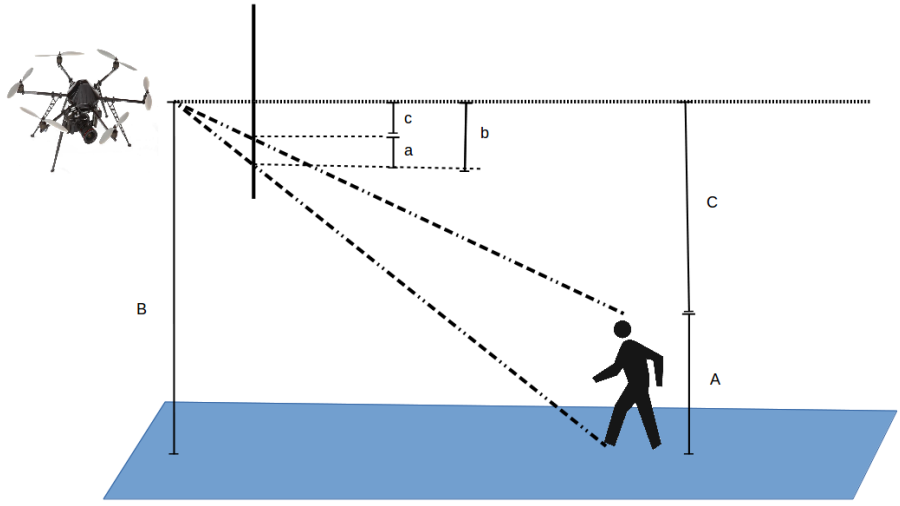


Figure 7.2: Side view of the scene with a forward-looking camera. The parameters shown on the figure are explained in the text, and used to create a first order linear function used as the ground constraint.

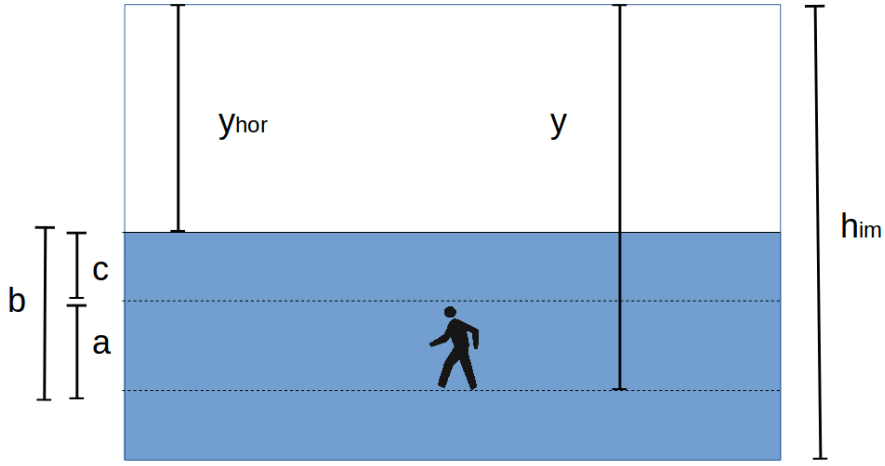


Figure 7.3: The image content as captured by the camera in the same scenario as figure 7.2. The parameters shown on the figure are explained in the text, and used to create a first order linear function used as the ground constraint.

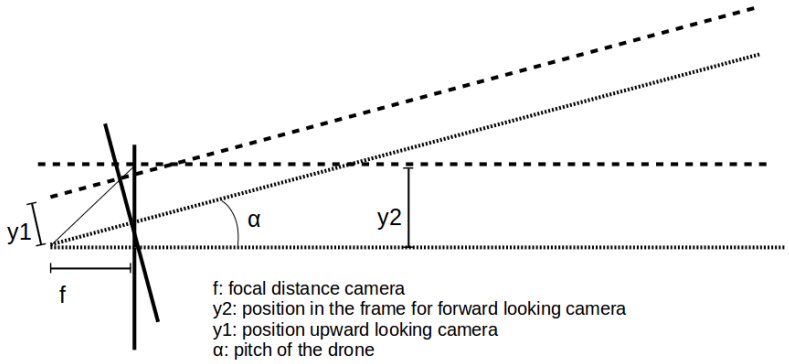


Figure 7.4: Compensate for the *pitch* by projecting a point to the forward-looking view. This figure is used to clarify the variables from equation 7.4.

When the pitch angle (α) can be measured, as in our UAV application, the solution to cope with this deviation is transforming the image back to a zero-pitch image.

Again using the pinhole camera model (as shown in figure 7.4), this transformation boils down to:

$$y_2 = f\frac{f.sin(\alpha) + y_1.cos(\alpha)}{f.cos(\alpha) - y_1.sin(\alpha)} \tag{7.4}$$

Based on the previous equations, we extended our ground plane constraint to cope with the extra degrees of freedom of a UAV. In section 7.7 we will evaluate this approach for our application.

7.4 Particle tracker

Although the pedestrian detection methodology we described in the previous section obtains good results, it does not form a reliable methodology on its own. Due to the variations in appearance a pedestrian can have, it is impossible to obtain a 100% reliable detection in each frame. Lowering the threshold will increase the recall and thereby the chance of detecting the pedestrian to follow, but will also increase the amount of false detections. Moreover, detecting at low thresholds requires a less strict rejection threshold used for the cascade, which implies less pruning during the detection pipeline, with an increased

computation time as a consequence. Evidently this is to be avoided for time-critical applications such as ours. Therefore we use a tracker to compensate for the missed detections.

Our particle filter implementation is based on the publicly available implementation of Kevin Schluff[1]. Each particle represents a state: $S = \{x, y, v_x, v_y, s\}$, where x and y represent the position of the particle, v_x and v_y represent the moving speed of the particle, and s represents the relative scaling from the initial object size.

A state-update is performed on each frame. The state of each particle is updated using a constant velocity transition matrix as given by equation 7.5. The new position of the object is determined by taking the weighted mean of the particles. The confidence (C) of each particle is calculated using equation 7.6, which uses the Bhattacharyya distance (D) between the colour histogram of the model and the colour histogram of the particle's position.

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{7.5}$$

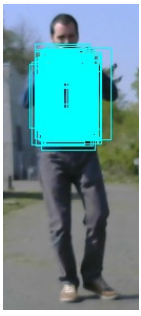
$$C = e^{-20.D^2} \tag{7.6}$$

To avoid the influence of background noise while tracking, we initialise the tracker not on the full detection rectangle, but on the chest (as visualised in figure 7.5). The chest does only move a little compared to for example the legs, and it is fairly easy to select an area on the chest which contains almost solely pixels belonging to the detected pedestrian.

For our application, we use 150 particles. Figure 7.6 contains some examples of the particle filter in action. Each rectangle represents a possible state of the target. The weighted average of all these samples forms the hypothesis we make of the current state of the pedestrian.

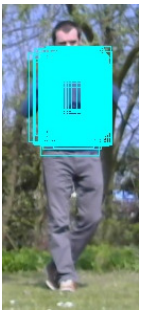
[1] https://bitbucket.org/kschluff/particle_tracker

Figure 7.5: The region on the pedestrian's chest we track, shown next to the manual annotation (in black) and the detection.



(a)

(b)



(c)

Figure 7.6: Visualisation of the particles while tracking.

7.5 PID control loop

The Proportional Integral Derivative (PID) control loop [11] is a commonly used system in many industrial applications. It uses an error measure, calculated as the difference between a measured variable and the desired value of that variable. By using this error value as a feedback, the PID system attempts to minimize the error over time by adjusting control variable. The output of the PID control loop is calculated by equation 7.7, for the discrete domain.

$$u(k) = K_p e(k) + K_i \sum_{0}^{k} e(k)\Delta t + K_d \frac{\Delta e(k)}{\Delta t} \tag{7.7}$$

In this equation, we recognise three terms, which each have specific function. The contribution of each of the terms is parameterized by the gain parameters K_p, K_i and K_d. The first term, the *Proportional* (P) term, measures the error directly as the difference between the measured variable value and the wanted value. The larger the error, the stronger the resulting value of the PID-loop, and as such, the stronger the correction will be (the faster the rotation of the UAV will be performed). The second term, the *Integral* (I) term, calculates the integral of the error values over time. When a residual steady-state error occurs in the system, this term will accelerate the steering to the correct position. The last term, the *Derivative* term, will slow down the speed of the steering when approximating the wanted variable value to avoid a (large) overshoot.

For our application we use a software based PID control loop to control the yaw steering of the UAV based on the error defined as the difference between the current position of the target (pedestrian) in the image and the centre of the image (which is the position with the lowest probability of loosing the pedestrian). In our application, we eliminated the use of the I-parameter, since we do not expect the presence of a continuous error while determining the position of the pedestrian.

The technique used to determine the P- and D-parameter is based on the "Ziegler-Nichols"-method [99]. Herein the D-parameter is initialized as zero. From then a number of iterations is performed in which first the P-parameter is incremented until the moment before oscillation of the error occurs, followed by incrementing the D-parameter to the same point. In the final setting, both the P- and D-parameter were set to 0.2. Note however that we use a scaling factor to ensure that the output of the PD-loop has the same value-range required for steering the UAV, which is based on the values normally received from the remote control.

7.6 Integration

7.6.1 System overview

Figure 7.8 gives an overview of the hardware we use. To demonstrate our framework we used the F550 hexacopter from DJI with the Pixhawk as stability controller. We equipped the F550 with the Brix computing module (Intel-I7 processor, 8G RAM and 100G HD with 172g of weight) to run our framework on. The choice of using the Brix is based on the tool developed by Hulens et al. [53]. For our application, the flight-time is estimated on 12 minutes when the algorithm runs at 15fps. As seen in figure 7.8, the Brix is communicating with the Pixhawk using the *Mavlink protocol*. Using this communication protocol, the sensor data including pitch, roll and yaw-angles as well as the altitude, are retrieved from the Pixhawk. To control the F550 from the Brix, several changes in the Pixhawk's firmware were made to receive Mavlink control messages from the Brix instead of from the remote control.

To combine all previous steps we developed a software framework, of which an overview is shown in figure 7.7. Each of these blocks is executed in a separate thread. The *Capture Frames* retrieves the frames from the camera. When the frame is retrieved, this thread requests the measured sensor-data from *Read Sensordata UAV*, which maintains communication with the Pixhawk using *Mavlink messages* as shown in figure 7.8. The *roll* is corrected by rotating the retrieved frame based on the measured sensor-data. Both the roll-corrected image and the sensor-data are passed to *Person Detection* and *Person tracking* using a queue.

The *Person Detection* performs the detection of pedestrians, using the ground constraint based on the sensor-data, as described in subsection 7.3.1. At the same time, the *Person Tracking* performs a state-update to obtain the most probable position of the person using colour-information based on 150 particles, as described in section 7.4. The results of both the person tracker and the person detector are passed to *Combine Results Control UAV*, which combines them to a single location. This is described in more detail in subsection 7.6.2.

The task of our UAV demonstrator is to steer the UAV such that a tracked pedestrian is kept in view, and therefore in the centre of the camera image. The difference between the desired position and the calculated position of *Combine Results Control UAV* is used to steer the UAV using a *PID control loop*. The latter calculates a smooth control value to steer the UAV, depending on the size of the error and the speed the errors changed in time. *PID control loop* maintains a continuous communication with the *Pixhawk*, which on his turn controls the motors of the UAV.

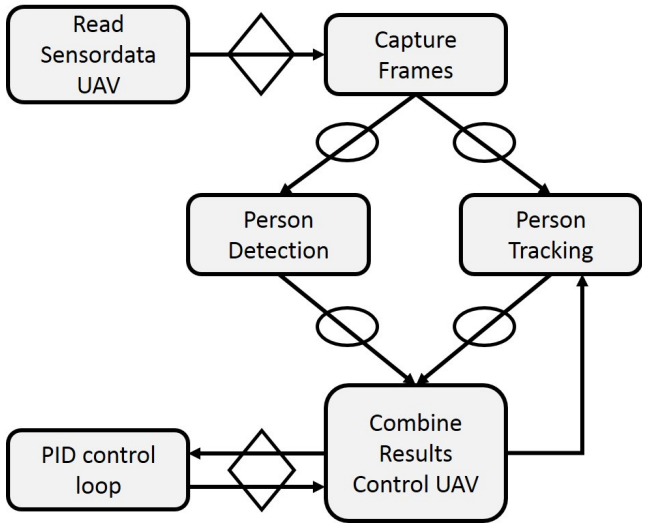


Figure 7.7: Overview of the UAV framework. Oval=queue, Diamond=mutex

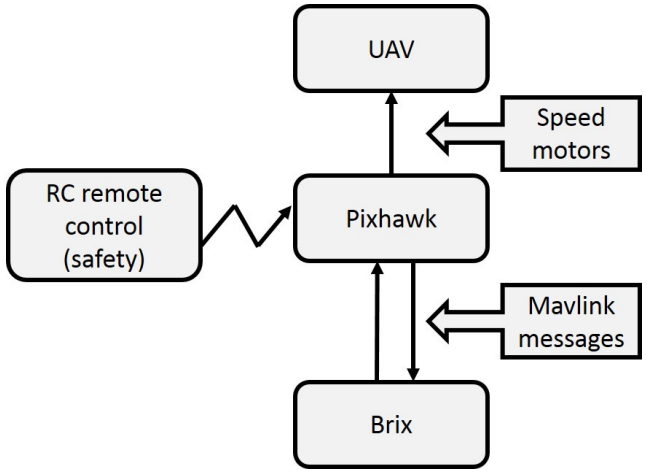


Figure 7.8: Overview of the communication between the UAV components.

7.6.2 Combination of pedestrian detection and tracking

We combine the detections of the pedestrian detector and the particle filter into a single hypothesis of the pedestrian's location. The tracker is controlled from the block *Combine Results Control UAV*. After the UAV takes off, the tracker is initialised on the strongest detection found. Hereby, the target pedestrian to follow is selected. From then on, the tracker is reinitialised using the pedestrian detections found by the pedestrian detector, which is performed in two cases:

1. When the tracking confidence drops below a 50% confidence, we search for detections with an overlap of at least 50% with the current tracking rectangle. The confidence score is calculated using the same method as for the particles, given by equation 7.6.
2. When the confidence of the tracker becomes very low we suspect the tracker to be drifted away. Therefore we calculate the confidence on all detections in the image. The tracker is reinitialised on the detection with the highest similarity between the detection box and the model.

In both cases, when no detection is found matching the criteria, no reinitialisation of the tracker is performed. Keep in mind we are currently focusing on tracking just one pedestrian, and this approach should be altered slightly to cope when multiple pedestrians are walking through the scene.

As we described in section 7.4 we do not use the original dimensions of a detections to initialise the tracker, but only a small rectangle focusing on the chest of the pedestrian for improved detection results. To perform the overlap criterion, we invert this operation to obtain the dimensions of the tracking rectangle as it would be for a full body tracking rectangle.

Note that we use the hypotheses of the particle filter as an input for the PID-loop, while we use the pedestrian detection algorithm to keep the tracker on the target. As long as the particle filter has a sufficiently high confidence, based on the aforementioned criteria, the tracker does not have to be reinitialised.

7.7 Experiments

In this section we discuss the experiments we have performed. We present the results we obtained when using the framework described in section 7.6 on a flying UAV. All speed results are obtained using the Brix embedded system.

Technique	Evaluation speed	Speed-up
Ours-ACF	14.9 fps	1×
Ours-ACF-groundconstraint-approximation	33 fps	2.2×
Ours-ACF-groundconstraint	21.9 fps	1.47×

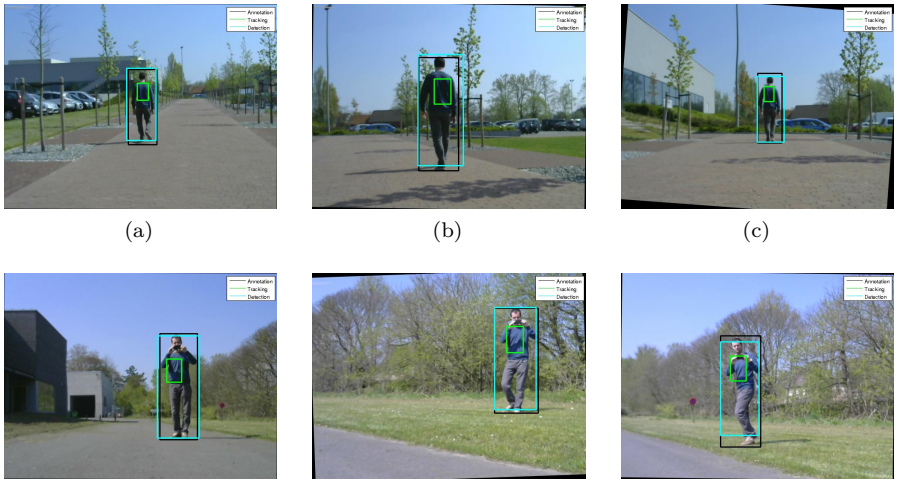
Table 7.1: Comparison of the detection speed between the different implementations used for the UAV application. The ground plane constraint is calculated based on the measured sensor data.

To evaluate the system we described in section 7.6, we performed a number of test flights. For two sequences, we will show detailed results. Example frames of these are shown in figure 7.9. The first sequence of 751 images shows the performance of our system while following a walking pedestrian from behind. The second sequence of 823 images keeps the UAV more or less at the same spot while following a pedestrian walking in front of the UAV. All the data (the frames, the tracked position and the detections) where saved to disk to allow comparison of different approaches. All the frames where manually annotated. Based on these sequences we will discuss the evaluation speed of the different detection options, as described in section 7.3, and the accuracy of the position determination of our system. We used a pedestrian detection model trained on the INRIA training set.

In table 7.1 we present the detection speed of our detector implementations. As can be seen, we reach decently high processing speeds on the on-board embedded system we use. Since we are using a 15fps camera, we chose the most accurate implementation, which uses the ground plane constraint without the approximation of features.

By adding the roll-angle to rotate the image, we make sure the frame can be processed while the pedestrians are straight up instead of rotated. This is visualised in figure 7.10. The use of roll-correction has a high impact on the amount of correct detections. To determine the influence of roll-correction, we annotated a dedicated experiment of 1367 frames. The amount of times the target pedestrian could correctly be detected without roll-correction is 115 times, or 8.4% of the images in the sequence. This is drastically increased to 692 times, or 50% of the images in the sequence when applying the correction. Since the detections are used to improve the tracking, and thus the steering of the UAV, this has a big influence on the resulting tracking performance.

It is important to keep in mind that the UAV application is composed from different techniques (pedestrian detection, using a ground constraint, a colour based particle filter which is kept in place with the pedestrian detection



(a) (b) (c)

(d) (e) (f)

Figure 7.9: Example images from the two sequences we use to validate our system: the tracked position at chest height, the annotation in black and the position as found by the pedestrian detector. Top row: sequence 1, bottom row sequence 2.

(a) Before roll-correction (b) After roll-correction

Figure 7.10: The visual effect of applying roll-correction on the captured frames.

Figure 7.11: We measure the "Fault" as the absolute pixel distance between the centre of the tracking hypothesis and the manual made annotation. According to equation 7.8 this value is normalised with the annotation's width.

algorithm, a software based PID control loop, ...) and it is important that each of these work properly to make the application work. The particle filter we use has some randomness though. To measure the accuracy of our combination of the ACF detector and the particle filter, we evaluate it, separately from the other components, 250 times on the frames of both our experiments. We measure the accuracy relative to the width of the annotation according to:

$$E = \frac{F}{W_{an}} \tag{7.8}$$

where F is calculated as the absolute pixel difference between the centre position of the tracking hypothesis and the centre of the manual annotations (the two vertical lines at the middle of the rectangles in figure 7.11), and W_{an} is the width of the annotation in pixels.

This means that as long as the error E is below 50%, we still steer the UAV to a position inside the annotation bounding box and so on the pedestrian. We obtain an average normalized error of *4.52%* and *5.64%* (averaged over our 250 independent runs) for experiment 1 and experiment 2 respectively, with a standard deviation of our normalized error value of 5.56% and 6.43%. Only 0.13% and 0.12% of the errors is above 25% with a maximum of 35% and 27%. So, although the particle filter introduces some randomness, despite the movement of the UAV, our system always determines the location of the

pedestrian in our datasets correctly (a location on the pedestrian), and in over 99,8% of the frames close to the centre. This is sufficiently accurate to steer the UAV towards the pedestrian, with the intervention of the PID loop.

During processing, the Brix embedded system has a workload of 95% and has a 22W power use.

	Sequence 1	Sequence 2
Detection	4.05%	4.59%
Tracking	4.52%	5.64%

Table 7.2: The normalised error, measured according to equation 7.8. Less than 50% means we are still tracking inside the annotation bounding box.

7.8 Conclusion

In this chapter, we implemented a system to automatically follow pedestrians with a UAV using an on-board embedded system. The requirement of performing pedestrian detection, among other tasks, in real-time on an embedded system required to select computationally efficient methods. Therefore we used *Ours-ACF*, which is both accurate and fast. For both speed and accuracy considerations, we reformulate the ground plane estimation technique we discussed in section 5.5, to be fully parameterizable by the sensor data measured by the IMU of the UAV. To overcome missed detections of the pedestrian detector, we use a colour based particle filter. Our complete system is successfully validated using a real flying UAV. Hereby we reach real-time processing on the on-board embedded system, while obtaining a normalised error of 5.56% averaged over 250 independent runs.

Chapter 8

Conclusion

In this dissertation we focused on the applicability of pedestrian detection algorithms in real-life applications. We created a framework of five pedestrian detection algorithms developed in C++, which allows a fair comparison. Based on a variety of datasets, we evaluated multiple techniques to improve the detection speed and detection accuracy of these detectors. The goal of this dissertation is to discuss techniques that improve the detection accuracy and/or speed, to benefit the applicability in real-life applications.

We presented a novel methodology to combine pedestrian detectors to improve the detection accuracy, which we coined *The Combinator*. This approach allows for a combination of an arbitrary number of pedestrian detectors. As the first step, we developed a technique to cluster detections based on an overlap criterion. For each of those clusters, we performed a combination rule, which takes a *confidence* and a *complementarity* measure into account. The *Confidence* measure indicates the performance of a detector individually, while the *Complementarity* measure indicates the additional value of the different detectors over each other. Based on our validation approach, we managed to find an optimal combination. We compared this approach with naive combination approaches, which shows that our combination methodology outperforms all of them, and shows that our approach is very competitive with the state-of-the-art.

To improve the detection speed of detectors, we proposed to use different methodologies, which can be combined. In a first methodology, we look into the use of multi-core architecture devices to improve the amount of data processed per time unit. We compare different techniques of using parallelisation, including evaluating the layers of the feature pyramid in parallel and evaluating multiple frames in parallel. Here we show the benefit of ensuring load balancing between

the threads, such that each thread has an equal amount of data to process. We also looked into the use of GPU hardware, which we demonstrated by implementing the feature pyramid of the DPM detector on GPU.

As a second methodology to improve detection speed, we looked into two methods to reduce the search space of the detection algorithm, by integrating scene knowledge. Note that these techniques require an adaptation to the scene, in contrast to the techniques based on multi-core architectures, which makes them less adaptable to other scenarios without changing parameters. Firstly we looked into the reduction of the reduction of the scale-space pyramid, by pruning scales we do not expect in a dataset. We demonstrated this approach using the Caltech dataset. Next, we looked into the use of a tracking-by-detection framework based on a Kalman tracker and the *Deformable Part Model* detector. We demonstrated this approach using three different settings and successfully match the accuracy of a full frame evaluation, but at a higher detection speed. As a final technique to reduce the search space, we looked into estimating the ground plane of the scene as a first order linear equation. This allows determining for each position in the image the expected size to find pedestrians in, within a certain margin. To decrease the search space, and by consequence improve the detection speed (and even accuracy by reducing the number of false detections), we use this calibration as a constraint, such that each scale should only be evaluated on a small area of the image.

Finally, we worked out two case studies where multiple of the previously described techniques are combined. Our first application focuses on the detection of pedestrians in the blind spot of a truck. This application requires both fast processing and high accuracy. We use the *Deformable Part Model* detector, which due to the allowed pose variation is preferred for this application. We combined a hybrid GPU/CPU implementation of this detector, which allowed fast processing on it's own, with the combination of a tracking-by-detection framework and a generalised ground plane constraint to reduce the search space to a minimum. To overcome the heavy image distortion coming with the wide-angle lens of a blind spot camera, we make use of the *Warping Window* technique.

As a second case study we developed an on-board implementation to follow pedestrians using a flying UAV. To improve both the detection speed and the accuracy of the *Aggregate Channel Feature* detector, we extended the ground plane estimation technique we noted earlier, such that it becomes parameterizable by the inertial sensor data of the UAV. To overcome frames without detections, we make use of a particle filter based on colour information, which uses the output of the detector to stay on the pedestrian. We empirically determined that we could sufficiently accurate determine the position of the pedestrian to steer the UAV with the help of a software based PID-loop.

8.1 Future work

In this section we discuss the future work of the research described in this dissertation. There is a lot of work that could build further on this research though. We have divided this in three subcategories, techniques that would improve the specific techniques of this dissertation, techniques that would improve pedestrian detection as a domain, and techniques that have to be improved for pedestrian detection to be better adaptable for real-life applications. Evidently will the improvements we give in the first two categories also be beneficial for applying pedestrian detection in applications.

8.1.1 Improving our techniques

The techniques we use as a baseline are accurate, but could however been improved to obtain an even higher accuracy. In [8], Benenson et al. give an overview of the evolution of pedestrian detection algorithms. They demonstrate the combination of multiple accuracy improvements on top of the already accurate *SqrtChnftrs* detector [7] to obtain higher accuracy. The techniques they use are *Local Decorrelation Features* [67], motion information [74] and a model to exploit the relation between two pedestrians [72]. As they demonstrate, these approaches are beneficial and complementary to other pedestrian detection algorithms, and thus potential techniques to also improve our own implementations. In [94], Yan et al. propose an extension of the *DPM* detector to take resolution differences of detections into account, reaching high accuracy. In future work, we could investigate the improvements these techniques could offer on top of the techniques we discussed in this dissertation. Important, however, is to keep an eye on at what computational cost these would come.

For *Channel based* detectors, we already have a very fast implementation at our disposal with the *ACF* detector. Since it already uses a very optimised implementation to calculate the channels, it may even be not worth to convert this to GPU. The part-based *DPM* implementation however, allows large room for improvement. In our implementation of section 4.7 we did not fully exploit the possibilities of the GPU hardware. This implementation will benefit from exploiting localisation by implementing the full pipeline in one kernel, such that each thread calculates the final feature values of a small section of the image. By avoiding the transfer to the global memory, which is currently the main bottleneck, a large speed-up could be obtained. The use of vector quantization on the HOG features, which is used in [80, 81], allows a large speed improvement without an accuracy loss. Note that the GPU implementation improves the

calculation time of the HOG features, while the vector quatization technique optimizes the model evaluation process.

In chapter 3 we proposed a technique to improve the detection accuracy by using multiple "less performing" algorithms as a combination. Especially when using this technique, it is important that each detector can be evaluated as fast as possible. The combination approach however is currently not fully explored and further research on this topic could lead to further accuracy improvements. Further improvements could possibly be obtained by the use of more dynamic values for *Confidence* and *Complementarity*, other combination methods, combinations of models trained on different datasets to avoid a training set bias, ... Based on our rule of thumb to use detectors that are very complementary, it could be beneficial to focus the training of detectors on this feature e.g. train a detector that especially focusses on finding pedestrians that are missed by the others, instead of finding all pedestrians.

The techniques we described in chapter 5 to reduce the search space based on scene knowledge, require currently a lot of manual input to configure. Although effective, these approaches would benefit from a more automatic system to determine a good setting.

In our tracking-by-detection framework of section 5.4.2, we use two parameters. The first is the location and size of *Initialisation Regions*, used to initialise tracks, while the second is the *lifetime* to overcome sequences where no matching detection can be found. These parameters are currently determined manually based on intuition and visual inspection of the data. However, both of these parameters could be determined automatically, based on an evaluation of the the pedestrian detector on full scale on a training dataset, and tracking each of the found pedestrians. The *Initialisation Regions* are determined as the locations where new pedestrians appear (such as the borders of the image), and regions of the image where pedestrian detection does not perform well, which could lead to lost tracks. The *lifetime* should be chosen such that tracks are allowed to recover from a sequence of frames where no matching detection can be found. Evidently a balance should be made between the number of regions (and their size) and the processing speed. Covering most of the image for full-scale detection, will induce a large accuracy at the cost of small speed improvement. Note however that these parameters are not independent, e.g. lowering the lifetime can induce more lost tracks such that more *Initialisation Regions* may be necessary, which makes finding the best parameter selection a challenging task. We suggest determining these parameters iteratively.

The ground constraint we use is currently determined as a first order function, which forms the relation between the y-position in the image and the height in pixels of the pedestrians. Although we used the ground truth for this task, this

can equally easily be obtained from detections. Important is that outliers should be pruned, and that the precision of the detector should be sufficiently high (avoiding false detections for this calibration) while still a lot of samples should be used. Note that during evaluation of the detector, the ground constraint could be updated based on the detections of the recent frames.

The UAV case we discussed in chapter 7 performs well for the application we had in mind, where we assumed certain constraints (single pedestrian, limited speed of movement, we could stay at eye-level of the pedestrian reducing the influence of viewpoint-induced distortions,...). Applying this in a more challenging context, such as following sports such as snowboarding, is however never tested and will probably require other settings to allow faster movement.

8.1.2 The pedestrian detection domain

Pedestrian detection has already made a huge progress in the last decade, especially in the field of accuracy. Currently a problem of the pedestrian detection techniques we use, is that the top performance is limited to a few datasets, and thus a limited context. This performance is therefore limited by the data it is trained on (training set bias). We experienced this when comparing the performance of ACF when trained on INRIA and Caltech in figure 5.14. This makes it a difficult choose which model to use in a real-life application. In the previous subsection, we already introduced the possibility of combining models trained on different datasets, but also there we use information of a single dataset to determine the *Confidence* and *Complementarity* values.

Although *Deep Learning* techniques are known to be able to generalise over a very large dataset, the computational cost is too high for sliding window evaluation. Therefore, in most cases a "traditional" pedestrian detection approach is used to propose a number of windows, which are then classified by the Deep Learning classifier. The same flaws of the baseline detector will also be present here.

Therefore a solution should be found to solve this bias, allowing to create a pedestrian detector that "perfectly" generalises the representation of a pedestrian in its model representation. A naive solution could be training a detector using the training data of multiple datasets, e.g. both INRIA and Caltech, but we would not expect that such a "general" detector will obtain the same performance as a detector that is trained specifically for that kind of images, but this should be tested to know for sure.

Since each detector is trained on only a sub domain (a limited dataset) of the problem domain (all possible scenarios with pedestrians), it seems impossible to generalise their performance of a sub domain to the whole domain. In the

future work of our combination approach [26], we suggested of determining the combination parameters based on features retrieved from the image, or even the selection of the detector to use for each window to classify. For example, if a detector performs well on high textured images, or high contrast, it can be assigned a higher confidence in a combination, or be the preferred detector of choice. Having information about the strenghts and weaknesses of all detectors can be of great help in selecting the "best" detector(s) for an application.

In our publication "Faster and more intelligent object detection by combining OpenCL and KR" [24], we proposed different levels of integrating Computer Vision and Knowledge Representation, a branch of Artificial Intelligence to describe and reason with knowledge. Knowledge about the scene can be greatly beneficial for scene understanding, which is not yet applied in literature. A first possible way of combining these two domains can be performed in a cascade, where first algorithms from the computer vision domain are used to retrieve data about the scene, which is then compared to a model, or mutual models, described in a knowledge representation language. For example, pedestrian detection and tracking can be used to obtain information about the players on a basketball court, which can then be interpreted by KR-models to find out which strategies are used, if violations against the rules are made, ... Note however that the correctness of this system depends on the accuracy of the computer vision algorithms used. For such an integration to work properly, a broad range of the most accurate computer vision algorithms need to be performed, with the corresponding computational requirements, such that the knowledge base has all the necessary (correct) information to work with. This flaw makes a cascaded approach challenging to work with in practical applications.

To meet this problem, a second possible manner of integration can be used, such that a bidirectional interaction exists between the two domains. For example, when a car is detected at a zebra crossing, based on a knowledge base a hypothesis can be made that a pedestrian should be found, that walks on this zebra crossing. It is however possible that the pedestrian detector does not find such a pedestrian, in which case the search can be repeated at a lower threshold or with a better performing pedestrian detector on a limited area (around the zebra crossing) to find support this hypothesis, or the hypothesis can be changed (e.g. maybe it is just a traffic jam). In this manner, a valid hypothesis can be made, and the most accurate algorithms, which are commonly the most computationally intensive, are only performed if needed. A correct interpretation of the scene can be beneficial for many applications. Another example can be a bank (or other company) with a security door. A person may only pass this door in certain circumstances, such as being staff, be accompanied by staff, cleaning staff during a limited time-window, ... These rules can be described using KR, which on its turn can trigger the appropriate algorithms

to validate a reason someone passed the door, using computer vision (face recognition for staff, tracking of pedestrians to obtain an optimal viewpoint for such recognition algorithms, ...). In this example, the computer vision algorithms are steered from KR.

Note that the information obtained as scene knowledge in knowledge representation, may also help to improve the accuracy of computer vision algorithms, since detections that do not comply with this information (e.g. the kind of distortions in the image, the size pedestrians have in relation to other objects,...) can be assumed incorrect detections.

8.1.3 Pedestrian detection in real-life applications

The goal of this dissertation is to improve the applicability of pedestrian detection for real-life applications. Although we presented multiple techniques that successfully improved both the accuracy and the detection speed, a guaranteed success in all circumstances is not reached. Real-life applications always have challenges, which we currently do not take into account. These could be as "simple" as occlusion or taking into account different appearances of the pedestrian such as falling down, which would require to use additional models for parts of the person [65, 85] or pose variations [95], but also as complex as fog or night time, which make the use of commonly used cameras, functioning mainly in the visual spectrum, unreliable.

Therefore we should look into the integration with other hardware that could reliably used in these situations, to overcome such flaws. One possibility is the use of an Infra Red (IR) camera, which allows using the body heat of pedestrians. But such hardware is not limited to cameras, e.g. ultrasonic distance sensors could also be very helpfull. An important criterion when selecting such hardware is the complementarity over other used techniques, such that flaws of one technology are solved by another. The output of such hardware could be combined to obtain a higher average detection performance [54], but should also benefit from a technique to detect the context (night time, fog, ...) that detectors that could not handle these circumstances are avoided. Currently we can find an integration of Infra Red cameras and visual spectrum cameras in literature [54], where the heat information is used as an additional channel in a *Channel Based* detector (ACF). But this is not the only, or maybe not even the best, way to combine multiple information sources. Training a single model based on all the information sources does not take into account that certain sensors will not perform well in all circumstances (such as visual spectrum cameras at night). Therefore it may be more beneficial to create multiple models, and combine them similarly to our *The Combinator* of chapter

3 technique, where the confidence depends on the context it is used in (day, night, fog, ...) such that the best working technique has the advantage. Techniques could also be implemented as a cascade. Similar to the technique of the *VeryFast* detector [6] where pedestrians are only searched for on elements perpendicular to the ground plane (*Stixels*) matching a pedestrian's height, pedestrians, or part-models when occlusion is taken into account, can be searched for only on locations that radiate sufficient heat, according to an IR-camera.

But also inside the possibilities of classic pedestrian detection, and our techniques, there are some flaws. For example, in the exploitation of the ground constraint we described in section 5.5, we calibrate a first order function based on the pedestrian height of adults. Although the accuracy improves over a complete dataset when applying the ground constraint by pruning false detections. This works for the test dataset since the larger portion of the annotations is in fact composed of adult pedestrians, almost no children are present. Note that this implies that pedestrians that do not cope with this constraint, including (small) children, will incorrectly be ignored. In the context of safety-critical applications, such as detection of vulnerable road users, this is a large safety risk! A possible solution to cope with this particular problem, is that we could use multiple calibrations, each targeting a specific pedestrian scale (range). Another possibility would be to use the scale of the pedestrian as part of a tracker's state vector.

Before using pedestrian detection techniques in practical applications, it should be validated on a dataset containing very diverse appearances of, and possible scenarios with, pedestrians. Each important type of occurrence (adults, children, cyclists, wheel chair users, elderly, people that have fallen down, ...) should be validated independently of the others under the applied constraints. This to ensure that all of these are sufficiently covered, and possible constraints do *not* form the risk of ignoring particular categories.

Bibliography

[1] Ahonen, T., Hadid, A., and Pietikäinen, M. Face recognition with local binary patterns. In *European Conference on Computer Vision (ECCV)*. Springer, 2004, pp. 469–481.

[2] Anthony, W. C++ concurrency in action. practical multithreading, 2008.

[3] Appel, R., Fuchs, T., Dollár, P., and Perona, P. Quickly boosting decision trees-pruning underachieving features early. In *JMLR Workshop and Conference Proceedings* (2013), vol. 28, JMLR, pp. 594–602.

[4] Bach, M. J. *The design of the UNIX operating system*, vol. 5. Prentice-Hall Englewood Cliffs, NJ, 1986.

[5] Benenson, R., Mathias, M., Timofte, R., and Van Gool, L. Fast stixel computation for fast pedestrian detection. In *European Conference on Computer Vision (ECCV)* (2012), pp. 11–20.

[6] Benenson, R., Mathias, M., Timofte, R., and Van Gool, L. Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR)* (2012), pp. 2903–2910.

[7] Benenson, R., Mathias, M., Tuytelaars, T., and Van Gool, L. Seeking the strongest rigid detector. In *Computer Vision and Pattern Recognition (CVPR)* (2013).

[8] Benenson, R., Omran, M., Hosang, J., , and Schiele, B. Ten years of pedestrian detection, what have we learned? In *European Conference on Computer Vision Workshop (ECCVW)* (2014).

[9] Benenson, R., Timofte, R., and Van Gool, L. Stixels estimation without depth map computation. In *International Conference on Computer Vision Workshops (ICCV Workshops)* (2011), IEEE, pp. 2010–2017.

[10] Benfold, B., and Reid, I. Stable multi-target tracking in real-time surveillance video. In *Computer Vision and Pattern Recognition (CVPR)* (June 2011), pp. 3457–3464.

[11] Bennett, S. A history of control engineering. *Control Engineering Series* (1930).

[12] Bourdev, L., and Brandt, J. Robust object detection via soft cascade. In *Computer Vision and Pattern Recognition (CVPR)* (2005), vol. 2, IEEE, pp. 236–243.

[13] Boyle, R. Belgium has nearly 300,000 surveillance cameras. http://www.xpats.com/belgium-has-nearly-300000-surveillance-cameras. Accessed: 2015-08-09.

[14] Bradski, G. Open computer vision library. *Dr. Dobb's Journal of Software Tools* (2000).

[15] Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., and Van Gool, L. Robust tracking-by-detection using a detector confidence particle filter. In *International Conference on Computer Vision (ICCV)* (2009), IEEE, pp. 1515–1522.

[16] Bron, C., and Kerbosch, J. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM 16*, 9 (1973), 575–577.

[17] Chandrakasan, A. P., Potkonjak, M., Mehra, R., Rabaey, J., and Broderse, R. W. Optimizing power using transformations. *Computer-Aided Design of Integrated Circuits and Systems (TCAD) 14*, 1 (1995), 12–31.

[18] Cho, H., Rybski, P., Bar-Hillel, A., and Zhang, W. Real-time pedestrian detection with deformable part models. In *Intelligent Vehicles Symposium (IV)* (August 2012).

[19] Dalal, N., and Triggs, B. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition (CVPR)* (June 2005), vol. 2, pp. 886–893.

[20] De Beugher, S., Brône, G., and Goedemé, T. Automatic analysis of in-the-wild mobile eye-tracking experiments using object, face and person detection. In *VISAPP* (2014).

[21] De Smedt, F., and Goedemé, T. Open framework for combined pedestrian detection. In *VISAPP* (2015), pp. 551–559.

[22] De Smedt, F., Hulens, D., and Goedemé, T. On-board real-time tracking of pedestrians on a uav. In *Computer Vision and Pattern Recognition Workshops (CVPRW)* (2015).

[23] De Smedt, F., Struyf, L., Beckers, S., Vennekens, J., De Samblanx, G., and Goedemé, T. Is the game worth the candle? evaluation of opencl for object detection algorithm optimization. *Perservative and Embedded Computing and Communication Systems (PECCS)* (2012), 284–291.

[24] De Smedt, F., Struyf, L., Beckers, S., Vennekens, J., De Samblanx, G., and Goedemé, T. Faster and more intelligent object detection by combining opencl and kr. *Journal of Ambient Intelligence and Humanized Computing (JAIHC) 5*, 5 (2014), 635–643.

[25] De Smedt, F., Van Beeck, K., Tuytelaars, T., and Goedemé, T. Pedestrian detection at warp speed: Exceeding 500 detections per second. In *Computer Vision and Pattern Recognition Workshops (CVPRW)* (2013), IEEE, pp. 622–628.

[26] De Smedt, F., Van Beeck, K., Tuytelaars, T., and Goedemé, T. The combinator: optimal combination of multiple pedestrian detectors. In *International Conference on Pattern Recognition (ICPR)* (2014), IEEE, pp. 3522–3527.

[27] Dollár, P. Piotr's Computer Vision Matlab Toolbox (PMT). http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html.

[28] Dollár, P., Appel, R., Belongie, S., and Perona, P. Fast feature pyramids for object detection. *Pattern Analysis and Machine Intelligence (PAMI) 36*, 8 (2014), 1532–1545.

[29] Dollár, P., Appel, R., and Kienzle, W. Crosstalk cascades for frame-rate pedestrian detection. In *European Conference on Computer Vision (ECCV)*. Springer, 2012, pp. 645–659.

[30] Dollár, P., Belongie, S., and Perona, P. The fastest pedestrian detector in the west. In *British Machine Vision Conference (BMVC)* (2010), vol. 2, Citeseer, p. 7.

[31] Dollár, P., Tu, Z., Perona, P., and Belongie, S. Integral channel features–addendum.

[32] Dollár, P., Tu, Z., Perona, P., and Belongie, S. Integral channel features. In *British Machine Vision Conference (BMCV)* (2009).

[33] Dollár, P., Wojek, C., Schiele, B., and Perona, P. Pedestrian detection: A benchmark. In *Computer Vision and Pattern Recognition (CVPR)* (June 2009).

[34] Dollár, P., Wojek, C., Schiele, B., and Perona, P. Pedestrian detection: An evaluation of the state of the art. *Pattern Analysis and Machine Intelligence (PAMI) 99* (2011).

[35] Dubout, C., and Fleuret, F. Exact acceleration of linear object detectors. In *European Conference on Computer Vision (ECCV)*. Springer, 2012, pp. 301–311.

[36] Enzweiler, M., and Gavrila, D. M. Monocular pedestrian detection: Survey and experiments. *Pattern Analysis and Machine Intelligence (PAMI) 31*, 12 (Dec. 2009), 2179–2195.

[37] Ess, A., Leibe, B., Schindler, K., , and van Gool, L. A mobile vision system for robust multi-person tracking. In *Computer Vision and Pattern Recognition (CVPR)* (June 2008), IEEE.

[38] Everingham, M., Van Gool, L., Williams, C., Winn, J., and Zisserman, A. The pascal visual object classes challenge 2009. In *2th PASCAL Challenge Workshop* (2009).

[39] Felzenszwalb, P., McAllester, D., and Ramanan, D. A discriminatively trained, multiscale, deformable part model. In *Computer Vision and Pattern Recognition (CVPR)* (2008), IEEE, pp. 1–8.

[40] Felzenszwalb, P. F., Girshick, R. B., and McAllester, D. Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/ pff/latent-release4/.

[41] Felzenszwalb, P. F., Girshick, R. B., and Mcallester, D. Cascade object detection with deformable part models. In *Computer Vision and Pattern Recognition (CVPR)* (2010).

[42] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. Object detection with discriminatively trained part based models. *Pattern Analysis and Machine Intelligence (PAMI) 32*, 9 (2010), 1627–1645.

[43] Fukunaga, K., and Hostetler, L. D. The estimation of the gradient of a density function, with applications in pattern recognition. *Transactions on Information Theory 21*, 1 (1975), 32–40.

[44] Garey, M. R., and Johnson, D. S. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing (SICOMP) 4*, 4 (1975), 397–411.

[45] Geiger, A. *Probabilistic Models for 3D Urban Scene Understanding from Movable Platforms*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2013.

[46] Geiger, A., Lauer, M., Wojek, C., Stiller, C., and Urtasun, R. 3d traffic scene understanding from movable platforms. *Pattern Analysis and Machine Intelligence (PAMI)* (2014).

[47] Girshick, R. B., Felzenszwalb, P. F., and McAllester, D. Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/ rbg/latent-release5/.

[48] Gove, D. *Multicore Application Programming: For Windows, Linux, and Oracle Solaris*. Addison-Wesley Professional, 2010.

[49] Grabner, H., and Bischof, H. On-line boosting and vision. In *Computer Vision and Pattern Recognition (CVPR)* (2006), vol. 1, IEEE, pp. 260–267.

[50] Hariharan, B., Malik, J., and Ramanan, D. Discriminative decorrelation for clustering and classification. In *European Conference on Computer Vision (ECCV)*. Springer, 2012, pp. 459–472.

[51] Hosang, J., Omran, M., Benenson, R., and Schiele, B. Taking a deeper look at pedestrians. In *Computer Vision and Pattern Recognition (CVPR)* (2015).

[52] Hulens, D., Goedemé, T., and Rumes, T. Autonomous lecture recording with a ptz camera while complying with cinematographic rules. In *Canadian Conference on Computer and Robot Vision (CRV)* (2014), IEEE, pp. 371–377.

[53] Hulens, D., Verbeke, J., and Goedemé, T. How to choose the best embedded processing platform for on-board uav image processing ? In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications.* (2015), IEEE.

[54] Hwang, S., Park, J., Kim, N., Choi, Y., and So Kweon, I. Multispectral pedestrian detection: Benchmark dataset and baseline. In *Computer Vision and Pattern Recognition (CVPR)* (June 2015).

[55] Kalman, R. E., et al. A new approach to linear filtering and prediction problems. *International Journal of basic Engineering (IJE) 82*, 1 (1960), 35–45.

[56] Khronos Group. Opencl - the open standard for parallel programming of heterogeneous systems, 2011.

[57] King, D. E. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research (JMLR) 10* (2009), 1755–1758.

[58] Kirk, D., and Hwu, W.-m. *Programming massively parallel processors: a hands-on approach*. Newnes, 2012.

[59] Kristan, M., Pflugfelder, R., Leonardis, A., Matas, J., Porikli, F., Cehovin, L., Nebehay, G., Fernandez, G., and Vojir, T. The vot2013 challenge: overview and additional results. In *Computer Vision Winter Workshop (CVWW)* (2014).

[60] Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems (NIPS)* (2012), pp. 1097–1105.

[61] Kuhn, H. W. The hungarian method for the assignment problem. *Naval research logistics quarterly 2*, 1-2 (1955), 83–97.

[62] Ladický, L., Sturgess, P., Alahari, K., Russell, C., and Torr, P. H. S. What, where and how many? Combining object detectors and CRFs. In *European Conference on Computer Vision (ECCV)* (2010), pp. 424–437.

[63] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal on Computer Vision (IJCV) 60*, 2 (Nov. 2004), 91–110.

[64] Lucas, B. D., Kanade, T., et al. An iterative image registration technique with an application to stereo vision. In *Internation Joined Conference on Artificial Intelligence (IJCAI)* (1981), vol. 81, pp. 674–679.

[65] Mathias, M., Benenson, R., Timofte, R., and Van Gool, L. Handling occlusions with franken-classifiers. In *International Conference on Computer Vision (ICCV)* (2013).

[66] Munshi, A., Gaster, B., Mattson, T. G., and Ginsburg, D. *OpenCL programming guide*. Pearson Education, 2011.

[67] Nam, W., Dollár, P., and Han, J. H. Local decorrelation for improved pedestrian detection. In *Advances in Neural Information Processing Systems (NIPS)* (2014), pp. 424–432.

[68] Naseer, T., Sturm, J., and Cremers, D. Followme: Person following and gesture recognition with a quadrocopter. In *International Conference on Intelligent Robots and Systems (IROS)* (2013), IEEE, pp. 624–630.

[69] Ojala, T., Pietikäinen, M., and Harwood, D. A comparative study of texture measures with classification based on featured distributions. *Pattern recognition 29*, 1 (1996), 51–59.

[70] Okuma, K., Taleghani, A., De Freitas, N., Little, J. J., and Lowe, D. G. A boosted particle filter: Multitarget detection and tracking. In *European Conference on Computer Vision (ECCV)*. Springer, 2004, pp. 28–39.

[71] Ouyang, W., and Wang, X. Joint deep learning for pedestrian detection. In *International Conference on Computer Vision (ICCV)* (2013), IEEE, pp. 2056–2063.

[72] Ouyang, W., and Wang, X. Single-pedestrian detection aided by multi-pedestrian detection. In *Computer Vision and Pattern Recognition (CVPR)* (2013), IEEE, pp. 3198–3205.

[73] Park, D., Ramanan, D., and Fowlkes, C. Multiresolution models for object detection. In *European Conference on Computer Vision (ECCV)*. Springer, 2010, pp. 241–254.

[74] Park, D., Zitnick, C. L., Ramanan, D., and Dollár, P. Exploring weak stabilization for motion feature extraction. In *Computer Vision and Pattern Recognition (CVPR)* (2013), IEEE, pp. 2882–2889.

[75] Pedersoli, M., Vedaldi, A., and Gonzalez, J. A coarse-to-fine approach for fast deformable object detection. In *Computer Vision and Pattern Recognition (CVPR)* (2011), IEEE, pp. 1353–1360.

[76] Pestana, J., Sanchez-Lopez, J. L., Saripalli, S., and Campoy, P. Computer vision based general object following for gps-denied multirotor unmanned vehicles. In *American Control Conference (ACC)* (2014), IEEE, pp. 1886–1891.

[77] PETS. Pets 2010 benchmark data. http://www.cvg.rdg.ac.uk/PETS2010/a.html, 2010.

[78] Prisacariu, V., and Reid, I. fastHOG - a real-time gpu implementation of HOG. Tech. rep., Department of Engineering Science, Oxford University, 2009.

[79] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)* (2014), 1–42.

[80] Sadeghi, M. A., and Forsyth, D. Fast template evaluation with vector quantization. In *Advances in Neural Information Processing Systems (NIPS)* (2013), pp. 2949–2957.

[81] Sadeghi, M. A., and Forsyth, D. 30hz object detection with dpm v5. In *European Conference on Computer Vision (ECCV)*. Springer, 2014, pp. 65–79.

[82] Siek, J. G., Lee, L.-Q., and Lumsdaine, A. The boost graph library. c++ in-depth series, 2002.

[83] Stix, V. Finding all maximal cliques in dynamic graphs. *Computational Optimization and applications 27*, 2 (2004), 173–186.

[84] Sudowe, P., and Leibe, B. Efficient use of geometric constraints for sliding-window object detection in video. In *Computer Vision Systems (CVS)*. Springer, 2011, pp. 11–20.

[85] Tang, S., Andriluka, M., and Schiele, B. Detection and tracking of occluded people. *International Journal of Computer Vision (IJCV) 110*, 1 (2014), 58–69.

[86] Van Beeck, K., De Smedt, F., Beckers, S., Struyf, L., Vennekens, J., De Samblanx, G., Goedemé, T., and Tuytelaars, T. Towards robust automatic detection of vulnerable road users: monocular pedestrian tracking from a moving vehicle. In *ATINER 7th annual international conference on computer science and information systems* (2011).

[87] Van Beeck, K., Goedemé, T., and Tuytelaars, T. A warping window approach to real-time vision-based pedestrian detection in a truck's blind spot zone. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO)* (2012), vol. 2, pp. 561–568.

[88] Vermaak, J., Doucet, A., and Pérez, P. Maintaining multimodality through mixture tracking. In *International Conference on Computer Vision (ICCV)* (2003), IEEE, pp. 1110–1116.

[89] Viola, P., and Jones, M. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition (CVPR)* (2001), vol. 1, pp. 511–518.

[90] Welch, G., and Bishop, G. An introduction to the kalman filter. 2006. *University of North Carolina: Chapel Hill, North Carolina, US* (2006).

[91] Willems, J., Debard, G., Bonroy, B., Vanrumste, B., and Goedeme, T. How to detect human fall in video. In *Positioning and Context-Awareness International Conference (POCA)* (2009).

[92] Xu, P., Davoine, F., and Denœux, T. Evidential combination of pedestrian detectors. In *British Machine Vision Conference (BMCV)* (2014).

[93] Yan, J., Lei, Z., Wen, L., and Li, S. Z. The fastest deformable part model for object detection. In *Computer Vision and Pattern Recognition (CVPR)* (2014), IEEE, pp. 2497–2504.

[94] Yan, J., Zhang, X., Lei, Z., Liao, S., and Li, S. Z. Robust multi-resolution pedestrian detection in traffic scenes. In *Computer Vision and Pattern Recognition (CVPR)* (2013), IEEE, pp. 3033–3040.

[95] Yang, Y., and Ramanan, D. Articulated pose estimation with flexible mixtures-of-parts. In *Computer Vision and Pattern Recognition (CVPR)* (2011), IEEE, pp. 1385–1392.

[96] Zeng, X., Ouyang, W., and Wang, X. Multi-stage contextual deep learning for pedestrian detection. In *International Conference on Computer Vision (ICCV)* (2013), IEEE, pp. 121–128.

[97] Zhang, H., Geiger, A., and Urtasun, R. Understanding high-level semantics by modeling traffic patterns. In *International Conference on Computer Vision (ICCV)* (2013).

[98] Zhang, S., Benenson, R., and Schiele, B. Filtered channel features for pedestrian detection. In *Computer Vision and Pattern Recognition (CVPR)* (2015).

[99] Ziegler, J. G., and Nichols, N. B. Optimum settings for automatic controllers. *American Society on Mechanical Engineering (ASME) 64*, 11 (1942).

List of publications

International conference publications

[1] Sander Beckers, Gorik De Samblanx, Floris De Smedt, Toon Goedemé, Lars Struyf, and Joost Vennekens, *Parallel sat-solving with opencl*, IADIS International Conference on Applied Computing, 2011, pp. 435–441.

[2] Sander Beckers, Gorik De Samblanx, Floris De Smedt, Toon Goedemé, Lars Struyf, and Joost Vennekens, *Parallel hybrid sat solving using opencl*, Benelux Conference on Artificial Intelligence (BNAIC) (2012), 11–18.

[3] Gorik De Samblanx, Floris De Smedt, Lars Struyf, Sander Beckers, Joost Vennekens, and Toon Goedemé, *Cpcpu: Coreful programming on the cpu: Why a cpu can benifit from massive multithreading*, Perservative and Embedded Computing and Communication Systems (PECCS), 2012, pp. 196–199.

[4] Floris De Smedt, Ive Billiauws, and Toon Goedemé, *Neural networks and low-cost optical filters for plant segmentation*, IADIS conference on computer graphics, visualization, computer vision and image processing, 2010, pp. 1–8.

[5] Floris De Smedt and Toon Goedemé, *Fast rotation invariant object detection with gradient based detection models*, VISAPP, vol. 2, VISIGRAPP, 2015, pp. 400–407.

[6] Floris De Smedt and Toon Goedemé, *Open framework for combined pedestrian detection*, VISAPP, 2015, pp. 551–559.

[7] Floris De Smedt, Dries Hulens, and Toon Goedemé, *On-board real-time tracking of pedestrians on a uav*, Computer Vision and Pattern Recognition Workshops (CVPRW), 2015.

[8] Floris De Smedt, Lars Struyf, Sander Beckers, Joost Vennekens, Gorik De Samblanx, and Toon Goedemé, *Is the game worth the candle? evaluation of opencl for object detection algorithm optimization*, Perservative and Embedded Computing and Communication Systems (PECCS) (2012), 284–291.

[9] Floris De Smedt, Tinne Tuytelaars, and Toon Goedemé, *Detection of abnormal behaviour in surveillance applications*, European Conference on the Use of Modern Information and Communication Technologies (ECUMICT) (2012).

[10] Floris De Smedt, Tinne Tuytelaars, and Toon Goedemé, *Detection of abnormal behaviour in surveillance applications*, European Conference on the Use of Modern Information and Communication Technologies (ECUMICT) (2014).

[11] Floris De Smedt, Kristof Van Beeck, Tinne Tuytelaars, and Toon Goedemé, *Pedestrian detection at warp speed: Exceeding 500 detections per second*, Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE, 2013, pp. 622–628.

[12] Floris De Smedt, Kristof Van Beeck, Tinne Tuytelaars, and Toon Goedemé, *The combinator: optimal combination of multiple pedestrian detectors*, International Conference on Pattern Recognition (ICPR), IEEE, 2014, pp. 3522–3527.

[13] Jan Reumers, Floris De Smedt, Jan Anthonis, Herman Ramon, and Toon Goedemé, *Indoor evaluation of crop row and grid detection-system for an automated transplanter.*, International Conference on Pattern Recognition Applications and Methods (ICPRAM), 2012, pp. 559–563.

[14] Kristof Van Beeck, Floris De Smedt, Sander Beckers, Lars Struyf, Joost Vennekens, Gorik De Samblanx, Toon Goedemé, and Tinne Tuytelaars, *Towards robust automatic detection of vulnerable road users: monocular pedestrian tracking from a moving vehicle*, ATINER 7th annual international conference on computer science and information systems, 2011.

Journal publications

[1] Floris De Smedt, Ive Billiauws, and Toon Goedemé, *Neural networks and low-cost optical filters for plant segmentation*, International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM) **3** (2011), 4.

[2] Floris De Smedt, Lars Struyf, Sander Beckers, Joost Vennekens, Gorik De Samblanx, and Toon Goedemé, *Faster and more intelligent object detection by combining opencl and kr*, Journal of Ambient Intelligence and Humanized Computing (JAIHC) **5** (2014), no. 5, 635–643.

Curriculum Vitae

Floris De Smedt was born in August 1986 in Lubbeek, Belgium. In 2007 he received his bachlor degree in electronics-ICT at Het Hoger Instituur der Kempen, Geel. In the context of his interest in network security, he made the thesis *De ontwikkeling van een honeypot, activiteiten van de cracker na de inbraak* (The creation of a honeypot, activities of crackers after the break-in.) at BA (Better Access), a company focussing on linux support and network security. In 2009 he obtained his Master degree for industrial engineering electronics-ICT. He made his masterthesis *Optimaliseren van positiemetingen op basis van randvoorwaarden* (Optimalisation of a position measuring system based on constraints.) in cooperation with Essensium, a spin-off company of Imec. After his studies, he started at the EAVISE research group at *Hogeschool voor Wetenschap en Kunst* in Sint-Katelijne-Waver (in the meantime, this campus has become part of the KULeuven), on the SIVOL (Snelle Implementatie van Vormgebaseerde Objectherkenning in Landbouwtoepassingen, *Fast implementation of shape-based object recognition in agricultural applications*) project, where he made his first steps in computer vision. At the end of 2010, he worked on the ICVS project, where his contribution was looking into the use of neural networks for on-loom textile camera inspection. In February 2011 he started his PhD on Pedestrian detection for real-life applications. During his PhD, he also worked on the *S.O.S. OpenCL* project where he looked into the data parallelisation opportunities of the novel (at the time) OpenCL language, and continued on the ICVS project where he looked into fault prediction for textile weaving.

Next to his interest in Computer Vision, Floris was an active volunteer member of Scouts Lubbeek for 19 years of which 8 years as a leader. He also has an interest in card magic, power training and indoor soccer.

FACULTY OF ENGINEERING TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
EMBEDDED ARTIFICIALLY INTELLIGENT VISION ENGINEERING (EAVISE)
Jan De Nayerlaan 5
B-2860 Sint-Katelijne-Waver
http://www.eavise.be